UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Telecomunicació
i Aeroespacial de Castelldefels

TREBALL DE FI DE GRAU

TFG TITLE: Numerical study with OpenFOAM of the bubble generation process in an inverse T-junction in microgravity conditions

DEGREE: Grau en Enginyeria d'Aeronavegació

AUTHOR: Gisela Rosado Bailón

ADVISOR: Santiago Arias

DATE: July 16, 2019

Títol: Estudi numèric amb OpenFOAM del procés generació de bombolles en una cruïlla en forma de T inversa en condicions de microgravetat

Autor: Gisela Rosado Bailón

Director: Santiago Arias

Data: 16 de juliol de 2019

Resum

La microgravetat és la condició que es percep en un objecte o cos quan l'efecte gravitatori és negligible. Això pot passar quan aquest es troba en un estat de caiguda lliure constant, el qual falseja una percepció de manca de gravetat que és molt útil d'estudiar per a aplicacions espacials o aplicacions a petita escala com la microfluídica.

En aquest projecte treballem amb simulacions numèriques per a estudiar la formació de bombolles en condicions de microgravetat, dins d'un capilar en forma de T. Les simulacions es fan a través de programari lliure com OpenFOAM, que s'encarrega de dur-les a terme; i ParaView, que fa el processament *a posteriori* i la visualització del problema. Dins d'OpenFOAM, el *solver* escollit és l'*interFoam*, ja que tracta el fluid multifàsic de les nostres simulacions com a incompressible, immiscible i isotèrmic sota un règim laminar.

Abans de generar resultats, la metodologia seguida a aquest treball implica generar la malla, aconseguir la formació de bombolles amb les condicions corresponents [1] i dur a terme tests de convergència de malla, pas de temps i angle de contacte a les parets del capilar. Als tests de convergència l'objectiu és utilitzar la malla més convenient possible per obtenir bons resultats i un pas de temps petit als càlculs per estar el més a prop possible de la convergència sense tenir un gran cost computacional. L'angle de contacte ens permet decidir quin dels fluids s'adhereix a les parets i com ho fa.

De cara als resultats, aquests es divideixen en dos parts. Per una banda, mesurem la pressió a quatre punts localitzats a l'afluència dels dos canals del capilar per a veure la seva evolució quan es forma una bombolla. I per l'altre banda, amb la malla, el pas de temps i l'angle de contacte escollits; combinem diferents velocitats d'entrada per ambdòs fluids per tal de veure com varien els paràmetres de les bombolles que es formen. La variació d'aquests paràmetres és el que ens fa determinar si el comportament de les bombolles és adequat i proper al comportament físic real.

En definitiva, la metodologia a seguir s'ha complert en tot moment, però algunes etapes d'aquesta s'han allargat o han presentat reptes que han arrossegat errors que han acabat perjudicant els resultats. Tot i així, donades les causes dels errors, els anàlisis de resultats i comportaments s'han pogut portar a terme de forma similar a l'esperada.

Title : Numerical study with OpenFOAM of the bubble generation process in an inverse T-junction in microgravity conditions

Author: Gisela Rosado Bailón

Advisor: Santiago Arias

Date: July 16, 2019

Overview

Microgravity is the condition perceived in an object or body when gravity effect is negligible. This can mostly happen when the object is in a constant free fall state, which gives the false perception of Zero Gravity. It being useful to study for space applications or small-scale applications like microfluidics.

In this project, we work with numerical simulations to study the bubble generation in microgravity conditions inside a T-junction channel. The simulations are made through open source softwares like OpenFOAM, which performs them; and ParaView, which is responsible of the post-processing and the visualization of the problem. The solver chosen from OpenFOAM is *interFoam* since it treats the multiphase fluid of our simulations as incompressible, immiscible and isothermal in a Reynolds laminar region.

Before generating results, the methodology of this project implies generating the mesh, get to detach bubbles with the correspondent conditions [1] and perform convergence tests for the mesh, the time step and the contact angle at the mesh walls. The goal of the convergence tests is to use the most convenient mesh possible to obtain good results and a small time step to be close to convergence without extra computational cost. The contact angle allows us to determine which fluid sticks to the wall the most and how this is done.

The results are divided in two parts. From one hand, we measure the pressure at four points around the junction of both pipes to see its evolution when a bubble is formed. On the other hand, with the chosen mesh, time step and contact angle; we combine different inlet velocities for both fluids to see how the bubble parameters change at their detachment. The change in these parameters makes us justify if the bubbles behaviour is adequate and close to the real physical behaviour.

In the end, the methodology has been followed at all times, but some stages of it have taken extra time or have presented challenges that produced larger errors in our results. Even so, despite the causes of these errors, the analysis of the results and behaviours have been made in a similar way as expected.

This is to my family, my colleagues
and my advisor Santiago Arias
for their support and help
through this project.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First, I would like to thank my advisor Santiago Arias for guiding me in this project. For the last seven and a half months I have immersed myself in this challenging project with no previous knowledge about multi-phase fluids in microgravity conditions. I have been a slow learner, especially with tools I have barely treated, but Santiago was patient with me and assisted me when I needed.

I enrolled myself in a numerical simulation subject in September 2018 to learn about numerical simulations in case I needed them for my final project, as Santiago advised me. I am really thankful for the opportunity that was given to me to do this project with him. I have been able to understand better the numerical simulations thanks to that subject.

I would also like to thank my colleague Adrià Espinosa for his help. He has been working with Santiago Arias as well in another project but using the same tools. I appreciate his time and tips to master OpenFOAM and Ubuntu, especially when we were both very busy with our respective projects. I appreciate it.

And last but not least, I would like to thank my family, my partner and my friends for being there every step of the way. I learnt a lot from this project, personally and professionally, and I would like to thank their patience and their faith in me to excel myself as a student.

CHAPTER 1. INTRODUCTION

In this project, we perform numerical simulations in which microgravity conditions play an important role. Before getting into detail and describing the aspects and features of the simulations, microgravity and its affects to scientific breakthrough nowadays must be understood.

1.1. What is microgravity?

The concept of gravity is very well known by the current society since Isaac Newton defined it more than three centuries ago. According to classical Newtonian mechanics, gravity is a force generated by the attraction of at least two masses, one being bigger and heavier than the others. This force is not constant and it weakens differently in each system, depending on the number of elements involved in it and their size.

An example of gravitational field could be a system composed by the Earth and a smaller body, like a human. In this case, Earth's mass is 10^{24} times bigger than a regular person of 80 kilograms, so it generates a force of attraction between the human body (the smaller mass) towards the centre of the planet. The reason behind a person jumping and an object falling to the floor is precisely that gravitational field.

The force of attraction created is directly proportional to the amount of mass difference between the bodies that conform the system. This one can be big enough to maintain the planets of the Solar System orbiting around the Sun (where the biggest mass is around $2 \cdot 10^{30}$ kilograms) or small enough to be neglected, considering the attraction between an apple and a TV.

Gravity usually weakens as the smaller mass distances itself from the largest. According to this principle, the further away a small mass gets from Earth in space, the smaller it is the gravity effect. That depends on the smaller mass, which may be big enough to orbit around Earth (like the Moon satellite) or small enough to keep an astronaut floating around the International Space Station (ISS). That floating is known as *Zero Gravity* effect. This effect is not real itself because gravity is never null unless the distance from one mass to the other is infinite, theoretically. Also, the term "floating" would not be accurate for the situation, since both the astronaut and the spacecraft are on a free fall with the same acceleration towards Earth. It is from the astronaut's point of view that this is seen as floating, even though gravity still works. What really happens is that the force of gravity remains high (gravity force at the ISS is nearly at 90% the Earth's surface value, [2]), but this one is perceived as if it was very small. When the perceived gravity acceleration is of about $10^{-6}g$ or less, the weightless effect appears. Also known as the condition of microgravity.

Microgravity is caused by an object in free fall or orbiting around a big mass. In some cases, it may also appear when the object gets far away from the gravitational source of interaction or when it is located at a point in space where the forces are in equilibrium. Only then, the object is said to be weightless, but in reality its mass remains the same.

1.2. Why is it important to study microgravity?

Many space agencies are interested in studying microgravity to understand what happens to an equipment, spacecraft, satellite or a living being in space. These study how the lack of gravity affects objects or systems that are originally from Earth. To do so, scientists and engineers have found alternative technologies to generate a weightless condition on Earth, as it is more affordable than going out space. The strength of the gravity force on the Earth's surface adds limitation in the time tests in these conditions. Nonetheless, these artificial conditions are still useful to study space matters before sending an equipment or people on a mission. The experiments reduce risks and increase the safety of those onboard.

The prestigious agency NASA has been studying for decades microgravity on Earth with the help of additional infrastructures. The facilities used for these studies allow the possibility to conduct experiments that last a few seconds or sometimes minutes. The research for new alternatives continues in order to find a way to experience weightlessness on free fall for a larger period of time. If this was endured, the missions on space would become safer and have a higher rate of success, thus everything would have been studied and tested on Earth already.

The drawback of all this is that the construction of a perfect scenario to conduct these experiments costs millions of dollars. So why do people invest in these, when there are so many other issues to fix in the world?

There is an article published in the digital Forbes magazine[3] which transcripts a response letter from the associate NASA director and top rocket scientist, Ernst Stuhlinger in the 1970s. This one justifies why investing in research and exploring space is non-negotiable, even when the population suffers.

A nun who had worked in Zambia wrote to the agency to ask *"how could they justify spending billions on the Apollo program when children were starving to death."*. Stuhlinger justified that NASA employees believed that the missions and travelling to the Moon or Mars in the future, would bring benefits and solutions for such grave situations on Earth on the long run. He compared this with the story of the microscope, which appeared after very serious plagues in Europe in the middle ages. Many people died without a cure, but after that the microscope helped many more people by eradicating lots of diseases.

The letter also explained how the American government worked in the 1970s. At that time, from the 200 billion dollars the president of the US used as a yearly budget, only 1.6% was given to space exploration in 1970. Stuhlinger ensured that the national budget distribution was planned with a year or so in advanced, and usually a percentage of it was allocated in foreign issues (if approved by the council), which could include aiding the poorest. Detect which countries needed the aid the most was easy thanks to satellites. This distinction could help with the budget's allocation decision for foreign affairs, and either way people had to be patient and wait for the benefits to come from future researches.

1.3. Studying microgravity on Earth.

This section is about the different kind of tests that can be made on Earth to test microgravity or weightless conditions.

1.3.1. Parabolic flights and drop towers

The most common way to create a microgravity condition, used by many organizations of the sector around the world like ESA in Europe, NASA in the US and JAXA in Japan; consists on making an airplane fly up-and-down parabolic manoeuvrers. The parabolic trajectory enables running some tests near a microgravity condition for a period of 20 seconds or so with an acceleration of $10^{-2}g$, usually. Figure 1.1 shows the trajectory of this facility.

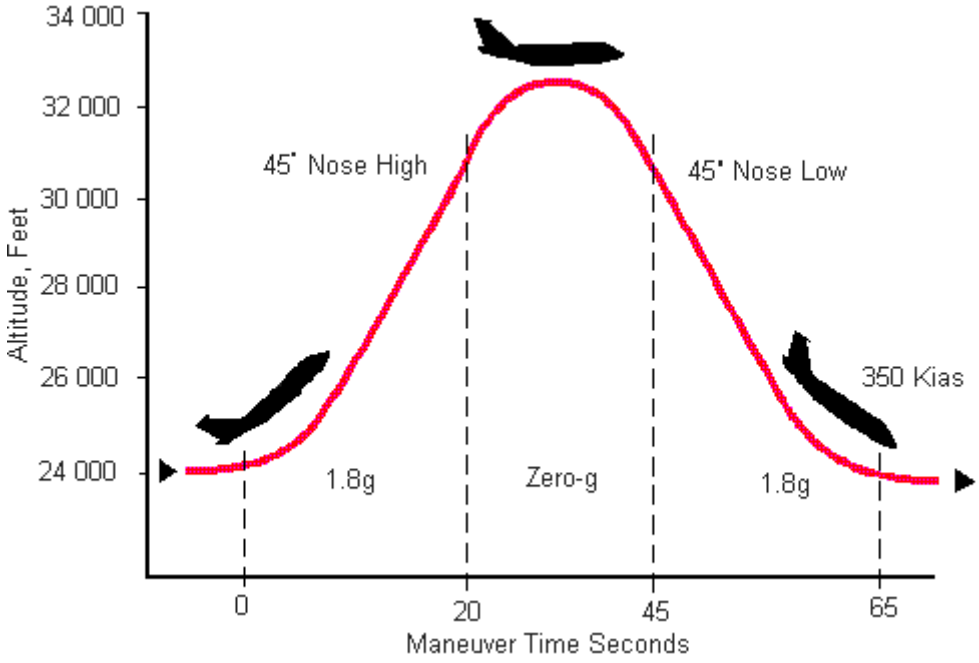


Figure 1.1: Parabolic pattern flown by the airplane to create microgravity condition [4].

There are similar facilities that improve the microgravity conditions perception to $10^{-5}g$ for tests and experiments in free fall and vacuum. NASA call them *Anti-Gravity Chambers*, but this type of facility is usually known as *drop towers*. Their structures are frequently made of steel or aluminium and have a cylindrical shape. The way they work is quite simple as it consists on dropping a chamber through the tower to experience a microgravity condition for a few seconds until the system brakes or impacts. NASA has two important drop towers: the *Zero Gravity Research Facility* and the *2.2 second drop tower*.

The first one is the worlds' biggest drop tower, which is about 142 metres deep and allows tests of around 5 seconds. This facility can study combustion, materials, biotechnology and fluid dynamics to improve future space missions technology.

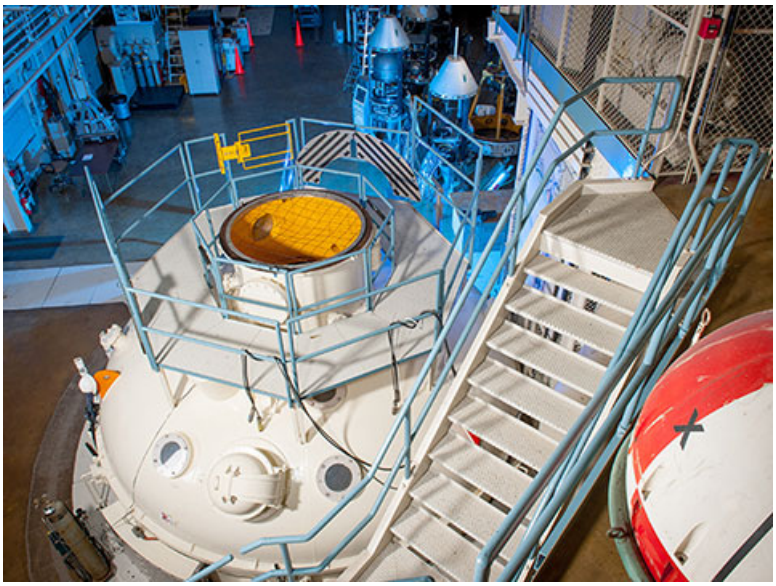


Figure 1.2: Drop Area of the Zero Gravity Facility [5].

The *2.2 second drop tower* performs tests of a couple of seconds at a distance of about 24 metres. It has been used for the last 50 years as a pre-test site for future experiments in the ISS. This drop tower uses a drag shield in order to minimize the aerodynamic drag in free fall to make the experiment as space alike as possible. The lack of drag implies putting an airbag at the end of the tower to reduce the impact of the object falling at such high velocities. The real acceleration is always 1g until the impact, where it can reach the 30-40g depending on the brake system.

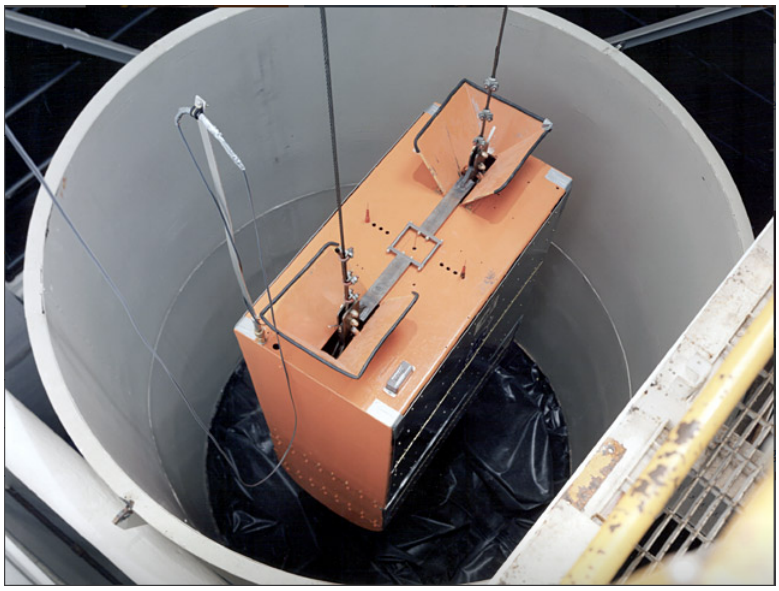


Figure 1.3: Drag shield and airbag of the 2.2 second drop tower [6].

Many universities around the world have collaborated in projects with space agencies to improve the test scenario created in the American company. This is the case of the European drop tower in the Centre of Applied Space Technology and Microgravity (ZARM),

from the University of Bremen. The project was a collaboration between the research centre and the European Space Agency, which implemented a new methodology for the drop tower by throwing the box upwards so that it can fall later on and increase the weightless condition time to 9.3 s.

Following the same work principle as the ZARM drop tower, the sounding rockets launch from 5 to 20 minutes to collect data and conduct experiments with a ”reduced cost”. The maximum height they can reach is of almost 1.300 km, and even though sounding rockets work like a drop tower, their flight trajectory is parabolic and the payload usually returns. When it does, it is gently dropped with a parachute as shown in Figure 1.4. These are the main mission developers of the NASA since 1959.

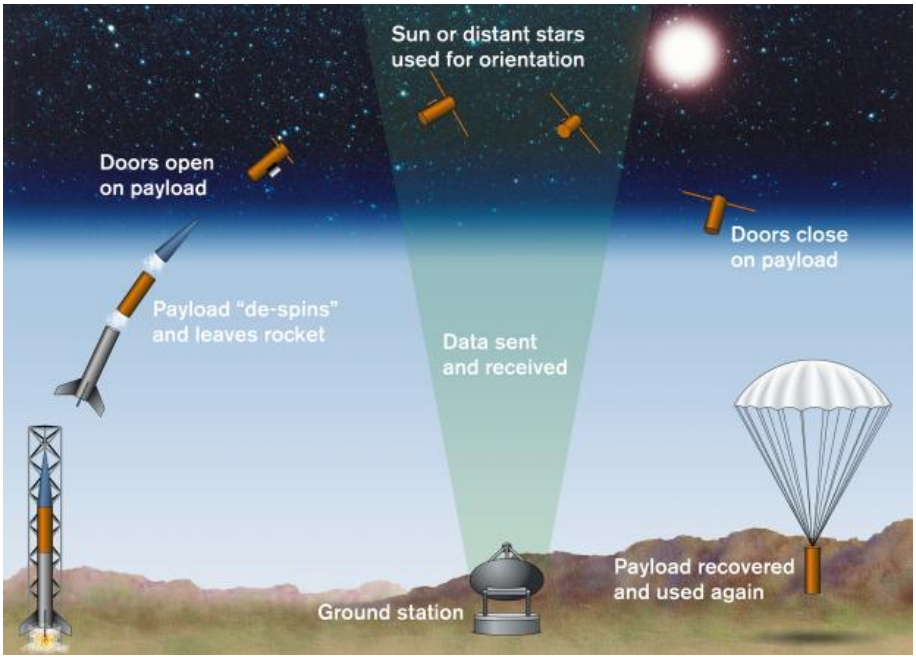


Figure 1.4: Sounding rocket’s flight trajectory [7].

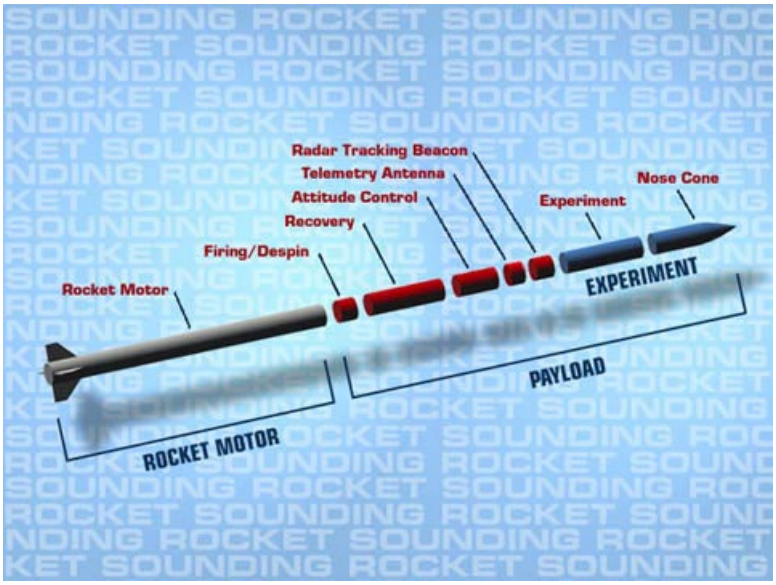


Figure 1.5: Sounding rocket’s structure [8].

To this day, NASA uses more than 15 types of sounding rockets from 2 to 20 meters long. All of them have the same phases though (see Figure 1.5). These are mainly used because of its time efficiency. The test time is much longer than a parabolic flight and a drop tower, so the 6 months of payload preparation tests are worth it.

JAXA works with one the most important drop towers in the world, MGLAB, and high-altitude balloons for tests. MGLAB tower gains 4.5 seconds of microgravity conditions with a maximum gravity acceleration of $10^{-5}g$, and its chamber is shown in Figure 1.6. As for the high-altitude balloons, these perform tests of 60 seconds long with a perceived gravity of $10^{3-4}g$.

Figure 1.6: JAXA drop tower chamber MGLAB [9].

After all, there are many ways to recreate microgravity conditions, even on Earth. Drop tower facilities present the best microgravity conditions, despite the alternatives. The effect of constant free fall can be experimented as well in the ISS or spaceships.

1.3.2. Space medicine and biology

When recreating microgravity facilities, its usage can be adapted to any area of knowledge. The human rational thinking does not work well in zero gravity conditions because we are used to the 1g Earthling scenario. Space medicine and biology are fields that can help us comprehend the living development in space and prevent people from getting hurt there.

Thanks to researches and systems like the Muscle Atrophy Research and Exercise System (MARES) from the European Space Agency, it is known that astronauts' bodies suffer changes due to the long exposure to weightlessness conditions. They loose the strength of the muscles and bones while the lack of gravity stretches their columns and makes them a few inches taller. It has been proved that the immune system acts in microgravity as if the body was infected, and it weakens after a long time activating defence mechanisms.

Other countries, such as Russia, have tried to expand the knowledge of these fields. They perform tests on Earth with humans exposed to weightlessness conditions, such as body immersions in water and head-down (6^{o}) anti-orthostatic hypokinesia experiments, among others. With these, the physiological effects of microgravity in human bodies can be no-

ticed before sending people to a mission. This type of experiments is also used by other space agencies, like NASA or ESA.

Figure 1.7: Example of a head-down (6^o) anti-orthostatic hypokinesia experiment [10].

Besides the tests on human bodies, missions have been sent to space with other ways of life like microorganisms, lizards, mice or seeds (BION M1 project, 2012) to see the effects microgravity have on them. They are currently working on a mission called *The Mars-500 project*, which started the microgravity tests on Earth in 2010 to prepare 6 males who will be sent to Mars some time in the following decades.

1.3.3. Fluids in space

The desire of exploring other planets involves studying everything that can be of use in them. Usually fluids are always present in their atmospheres, so many agencies study how to treat any kind of fluid in space for that purpose.

The China Space Agency, for example, is specialized in studying different fluids, the heat transfer process and complex fluid mechanics on space. This agency and other organizations from around the world like NASA, study together the fluids in the ISS to predict their behaviour to prevent accidents that may occur in spaceships or inside the ISS. There are plenty of fluids in space that behave contrary to human's logic based on Earth's gravitational field, and they may cause damages through an undesired combustion or scape, for example. Studying the way fluids escape or burn in space may help prevent an incident in satellites, spaceships or stations.

The study of gas combustion on space revealed that fire behaves differently as the flames become more round than they would in 1g. The lack of gravity makes the difference between the densities of combusted air and its surrounding air spread uniformly. Then, a layer at the boundary of the flame is formed where the oxygen and the fuel meet to start the flame. This is shown in Figure 1.8.

The peculiar behaviour from the gas combustion on space is one of the reasons why microgravity should be studied: because humans cannot figure this out by themselves without proof that help our understanding. Also, in case of a fire onboard, this helps astronauts

to be prepared and well-aware of how to extinguish it. The lack of awareness in this case could cost their lives.

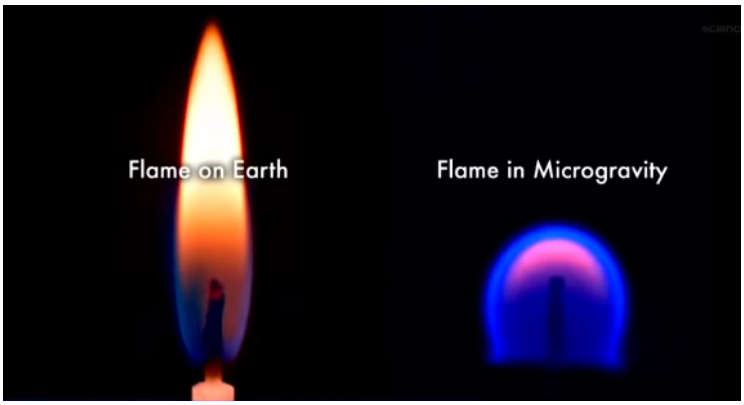


Figure 1.8: Comparison between a flame under gravity and microgravity effects [11].

1.3.4. Low-cost microgravity facilities

Being aware of the benefits the study of microgravity brings to society may be the main cause to invest in space researches, but some companies have tried to fight the high costs of these to enable a wider range of tests possibilities in the sector; as not many of them can afford using, for example, one of the NASA facilities.

In Europe, initiatives to create low-cost microgravity facilities have arisen. Near the campus, the HEMAV foundation is currently working on one of those to run some tests in weightless conditions with drones. The name of the prototype is GRAVIMAV[12] and it should carry a maximum payload of 10 kilograms per test. A student from the aerospace faculty of Castelldefels initiated this with his final project[13], where he tried to rotate the direction of the velocity axis on drones so that these could accelerate themselves downwards and create a little free-gravity fall for a couple of seconds. As far as it is known, this is still work in progress.

Other affordable ways to recreate microgravity are studies supported on open-source simulation softwares. This is the method we use in this project.

1.4. Microfluidics

Apart from the experimental platforms explained in 1.3., other scenarios where gravity is neglected can take place on Earth without the need of extra facilities. This is what happens in microfluidics.

Microfluidics is the science that involves technologies and systems to deal with small quantities of fluid at the sub-millimetre scale, inside hollow microfluidic channels. At such scale, gravity still exists on Earth's surface (1g) but it is neglected in the study because it is a much minor force. Then, the forces that remain strong take over and control the process. In our simulatons, this is what capillary forces do.

The applications of this science are very diverse, and some of them are present in very

mundane fields like medicine or pharmacy. There are many microfluidics studies focused on the generation of tiny capsules or the research for new cures for mortal diseases.

The most common manipulation of fluids in microfluidic channels is the synthesis of drugs to create the medicines that people buy on pharmacies. All capsules need a determined quantity of substance inside, so a distribution system of sub-millimetrical scale must be used to fill each capsule with the correspondent quantity of fluid. These distribution systems can be used in research as well to determine the behaviour of some cells to certain drugs, or to separate substances. The technologies used for these distribution systems are called *lab-on-chip technologies*, as they are used mainly on labs and research studies.

One of the latest examples of microfluidics applications is based on a tiny labyrinth chip that passes samples of blood through its channels. These channels contain compounds like drugs or medicines to see how they affect the sample, which contains cancer cells in it. A similar chip has been used at the University of Michigan to separate the cancer cells from the samples. Figure 1.9 shows the blood sample running through the channels of the chip.

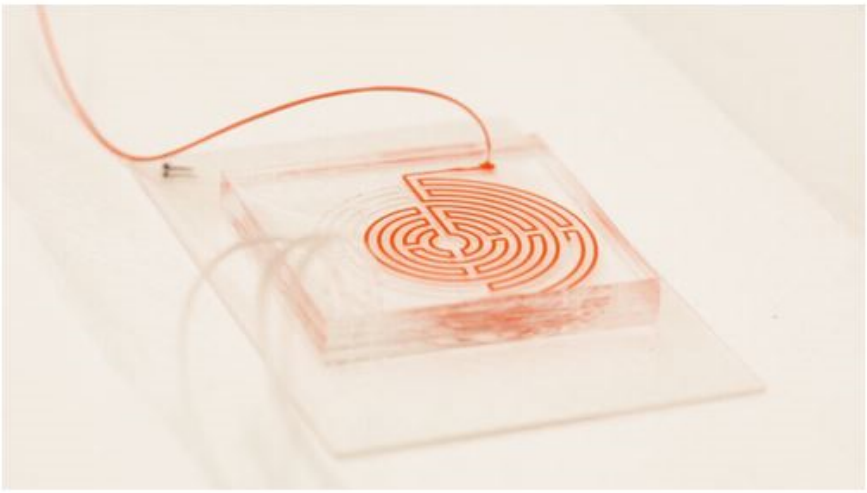


Figure 1.9: Lab-on-chip system for cancer cells research purposes [14].

1.5. Motivation of this project

In this project, we are going to perform numerical simulations of multiphase fluids that consist on one liquid and one gas. This combination is quite common in space studies, as space agencies of many countries like China investigate its behaviour. The fluids used are air and water flowing through very tiny channels with squared section. Those channels are part of our mesh, which recall typical scenarios from microfluidics studies.

In our simulations, a set of bubbles form from the air and water interaction. The goal is to understand how this interaction is produced when gravity effects are neglected and the capillary forces are dominant.

CHAPTER 2. PROBLEM STATEMENT AND METHODOLOGY

In this chapter, explanations regarding the physical problem treated and the necessary elements to run the numerical simulations will take place.

At the end of this chapter there will be a little introduction about the post-processing of chapter 3.

2.1. Background

The main goal of this project is to study through numerical simulations the bubble detachment in a T-junction pipe. Initially, the T-junction was supposed to be cylindrical to compare our simulation results with the experimental ones published in Arias and Montlaur(2017)[1]. Parameters, mesh convergence and initial and boundary conditions will be applied the same way as specified in [1], with just a few adjustments.

A gas and a liquid fluids should be injected from one inlet each, creating a T-shape between both fluids. The fluids used for the simulations are air and water, according to [1], and their volume average void fraction (α) is assigned as 1 and 0, respectively.

The scenario used for the simulations in this project has switched inlets from the experiments in [1], as shown in Figure 2.1. Nevertheless, the bubble detachment should take place at the same area where both fluids collide. When the bubbles move along the pipe, the computations should lead to a similar conclusion.

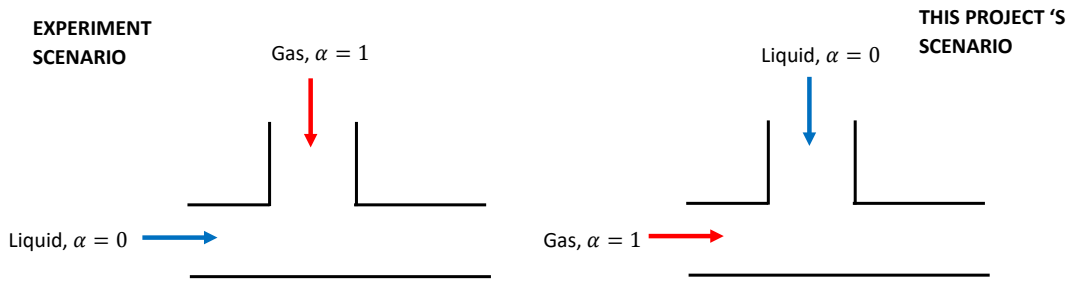


Figure 2.1: Difference between [1] scenario (left) and project's scenario (right).

Figure 2.2 and the following list provide this project's tasks and the time dedicated to each one. These do not include the report related work.

1. Install the required programs, virtual boxes and operating systems.
2. Learn how the programs work through tutorials.
3. Start the mesh generation process with a coarse mesh.
4. Introduce the initial and boundary conditions.

5. Introduce the control commands of the simulation and the rest of parameters.
6. Test the bubble detachment.
7. Introduce probes and samples in the simulations.
8. Generate post-processing codes in Matlab to process the collected data.
9. Modify simulation conditions or code, if needed.
10. Run convergence tests.
11. Simulations with different inlet velocities for results.

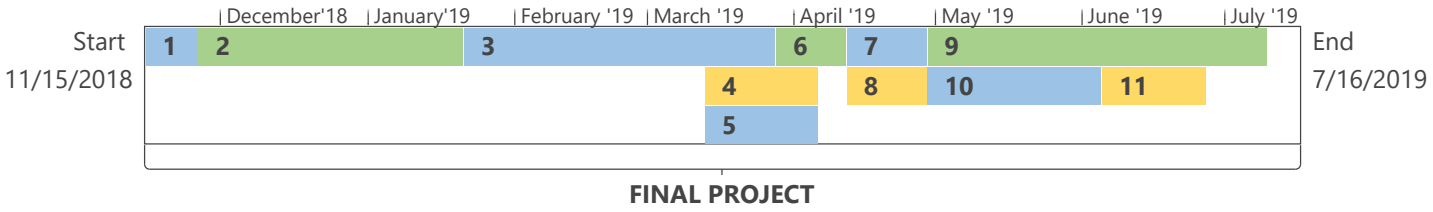


Figure 2.2: Time scheme for project's planned tasks[1].

2.2. Problem statement

This section exposes all the physical aspects regarding the simulation, from fluid parameters to all the conditions that define the work regime of the problem.

2.2.1. Fluid characteristics

The problem statement is based on the injection of air and water through two inlets in a T-junction pipe. When the fluids are injected, these mix, create some bubbles if the fluid characteristics let them, and then leave through one outlet.

Gravity is not considered in these simulations, so the balance between viscous forces and the surface tension at the layer between both fluids is what determines the detachment of any bubble.

The main properties of air and water depend on the temperature of the simulation scenario. The simulation itself is isothermal with a room temperature of 25 °C. According to this, standard physical values are used for both fluids. The parameters use a sub-index G for air (G from gas) and L for water (L from liquid), as shown in 2.1.

[1] Diagram made in *Microsoft Project 2016*.

Property	Gas value	Gas symbol	Liquid Value	Liquid symbol	Units
Viscosity (μ)	10^{-5}	μ_G	10^{-3}	μ_L	$Pa \cdot s$
Density (ρ)	1.225	ρ_G	10^{3}	ρ_L	kg/m^3
Kinematic viscosity (ν)	$8.163 \cdot 10^{-6}$	ν_G	10^{-6}	ν_L	m^2/s
Sigma (σ)	0.072				N/m

Table 2.1: Basic fluid parameters for air and water at 25^oC room temperature.

The parameters in 2.1 are adopted from [1] and [15]. The kinematic viscosity is computed dividing the viscosity by the density. This is made per each fluid as it determines if the fluids are viscous enough to create a bubble. If the kinematic viscosity is confused with the viscosity itself, the bubbles never form since the fluid would be perceived in simulations as too viscous. The viscosity (μ) is the only parameter that does not appear in the definition of the transport properties or work regime of the simulations.

2.2.2. Work regime

Air and water are considered incompressible since water is a liquid and a low-speed gas belongs to a subsonic regime, which at such small scale can be considered as incompressible, too. These flow at low speeds in this project, so their inlet superficial velocities must be smaller than $1\,\mathrm{m/s}$. Gas and liquid superficial velocities are 2.1 and 2.2.

$$U_{SL} = \frac{Q_L}{A} \tag{2.1}$$

$$U_{SG} = \frac{Q_G}{A} \tag{2.2}$$

Being,

Q_L = Liquid volumetric flow rate at the inlet

Q_G = Gas volumetric flow rate at the inlet

A = Capillary cross-section of each inlet

The inlet velocities define important parameters for our simulation. One of those is the Courant number, which must be as close to 0 as possible to compute correctly. The Courant number shows the relation between the cell size of the mesh (Δx) and the time step of the computations (Δt), while also depending on the gas velocity at the inlet (U_{SG}). That dependency is exclusive of this project.

$$Co = \frac{U_{SG} \cdot \Delta t}{\Delta x} \tag{2.3}$$

The Bond number shows the equilibrium between the gravitational and the capillary forces, which are strongly related to the surface tension defined in 2.2.1.. This number is defined in 2.4.

$$Bo = \frac{(\rho_L - \rho_G) g \cdot \phi_c^2}{\sigma} \tag{2.4}$$

Where,

g = Gravity

ϕ_c = Capillary internal diameter (around 10^{-3}m in this problem)

The equation 2.4 shows Bond number's dependency from gravity. It must be smaller than a threshold of 0.29 ($Bo < 0.29$) in order to consider gravity effects negligible [1]. If gravity is considered as zero (g = $0\,m^2/s$), the Bond number stays below the threshold for sure. Otherwise, the condition should be checked every single time a simulation finishes.

The dimensionless Weber number defines the work regime as well. It depends on the bubble velocity (U_G) and it can be defined for both fluids through equations 2.5 and 2.6.

$$We_{SG} = \frac{\rho_G \cdot \phi_c \cdot (U_G)^2}{\sigma} \tag{2.5}$$

$$We_{SL} = \frac{\rho_L \cdot \phi_c \cdot (U_{SL})^2}{\sigma} \tag{2.6}$$

$$U_G = C_0 \cdot (U_{SG} + U_{SL}) \tag{2.7}$$

Where,

C_0 = Void fraction distribution coefficient.

The Weber number evaluates the competition between the capillary and the liquid drag forces at the interphase of two multiphase flows[1]. The desirable Weber must be below a threshold value of 2 ($We < 2$) so that the capillary forces overcome the inertial ones [1]. If this is accomplished, the bubble detachment will take place.

The number of Reynolds (Re) is important to specify the turbulent state of the simulation's flow. There are two types of Reynolds that can be defined in this project: Re_M for average mixture superficial velocity, and Re_L for liquid superficial velocity. The formula is coincident for both Reynolds, but the Re_M takes into account the sum of inlet velocities (U_M from equation 2.9), while Re_L considers the liquid inlet velocity (U_{SL}).

$$Re_{M/L} = \frac{\rho_L \cdot \phi_c \cdot U_{M/SL}}{\mu_L} \tag{2.8}$$

$$U_M = U_{SG} + U_{SL} \tag{2.9}$$

Lastly, the Capillary number (Ca_{SL}) is defined as:

$$Ca_{SL} = \frac{We_{SL}}{Re_{SL}} \tag{2.10}$$

The first set of simulations have both inlet velocities set to $0.1\,m/s$ and its corresponding velocity for the fully developed fluid is $0.4\,m/s$ (section 3). According to these, the dimensionless numbers that define the simulation work regime are presented in 2.2.

Parameter	Bo	Co	C_0	Re_M	Re_L	We_{SL}	We_{SG}	Ca_{SL}
Value [-]	0	0.01 to 0.06	2	200	100	0.14	$2.72 \cdot 10^{-3}$	$1.39 \cdot 10^{-3}$

Table 2.2: Work regime dimensionless parameters.

According to table 2.2, the Bond and both Weber numbers are below their limit values, the Courant is very close to 0 empirically and the Capillary number is the same order as in [1]. That means the simulation works just fine and the capillary forces overcome the inertial ones to form bubbles. Also, both Reynolds are below 2,300, which is the turbulent flow threshold [16]. This means that our simulations correspond to a Reynolds in laminar flow conditions region.

2.3. Pre-processing

All the software involved in this project will be explained in this section, along with the definition of initial and boundary conditions in it and the post-processing program management.

2.3.1. Software and computer requirements

Free source software will be used for the computations and visualization of the simulations. This presents the advantage of being available for anyone, including students. However, this kind of programs have a less intuitive approach.

In this project, OpenFOAM is the program used for computations and ParaView does the post-processing and visualizations. The versions installed have been the 6.0 for OpenFOAM and the 5.4 for ParaView. These two are meant to be used in Ubuntu operating system as they work through commands. Windows is the default operating system of the local PC, so a virtual box has been installed to create an Ubuntu 18.04 platform to work on. The virtual box used is *Oracle VM Virtual Box*.

Matlab 2019a is installed in Ubuntu as well to do the post-processing of the simulations collected data. Some of the mesh generators were in it too. However, this report is not written in Ubuntu but in LaTeX, which is installed in the operating system of Windows.

2.3.2. Mesh generation

This section explains the process of mesh generation, along with some of its difficulties and adjustments. Further details about this section are available in B.

2.3.2.1. Geometry

The geometry of the mesh has been changed due to the difficulties presented while creating the initial mesh. At first, the geometry of the mesh was based on a cylindrical T-junction (see Figure 2.3).

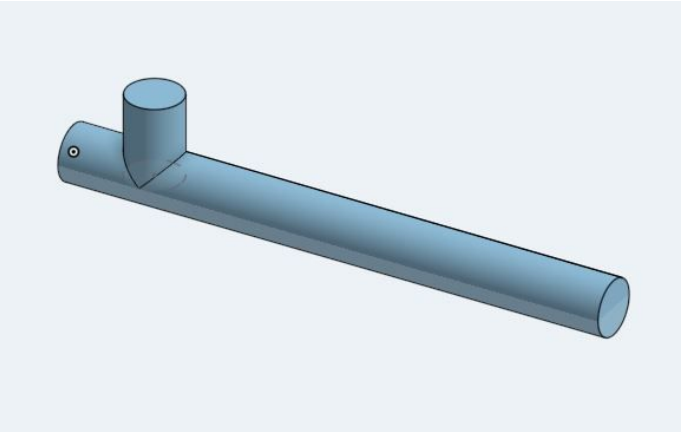


Figure 2.3: Initial mesh geometry created in [17].

The dimensions of the initial mesh are shown in Figure 2.4 in millimetres.

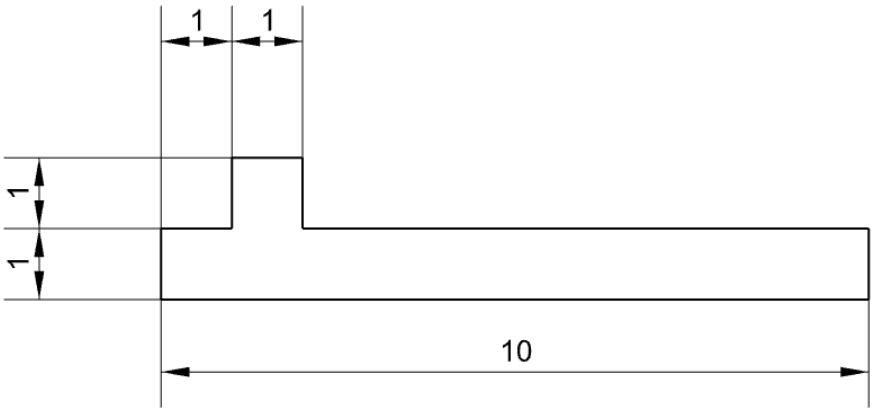


Figure 2.4: Mesh dimensions in millimetres[2].

Even though Figure 2.4 shows the real dimensions of the mesh, at the beginning to test boundary conditions and the bubble detachment, a shorter mesh was used to focus on the joint. This one was 5 mm long and its vertical pipe was reduced to 0.5 mm, just like in 2.5.

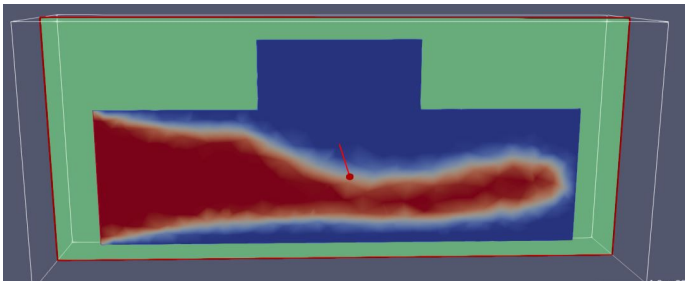


Figure 2.5: Example of short mesh for initial simulations[3].

[2]Dimensions scheme created in *NX Siemens 10.0* modelling software.
[3]Caption extracted from ParaView simulation visualizer.

The newest geometry maintained the dimensions from both the regular and reduced meshes. The only thing that changed was the section of both pipes, which were now squares of 1 millimetre per side (see Figure 2.6). With it, the problem simplified itself, so extra work is added to compensate the drawback.

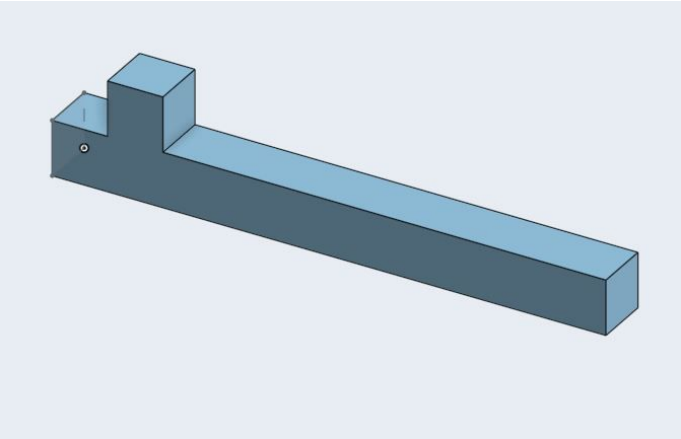


Figure 2.6: Final mesh geometry created in [17].

When generating a mesh, the walls had to be identified as face groups (or patches) to assign the boundary conditions to each of them. For the final mesh, 5 types of face groups are differentiated: *gasInlet*, *liquidInlet*, outlet, *walls1* and finally, *walls2*.

The *gasInlet* corresponds to the injection of air in the horizontal pipe, while *liquidInlet* is the patch associated to the vertical pipe inlet. The *outlet* patch is located at the end of the horizontal tube, and then there is the distinction between *walls1* and *walls2*. *Walls1* includes two vertical walls covering the vertical pipe, while *walls2* cover the walls forming the horizontal pipe. Figure 2.7 shows the patches scheme.

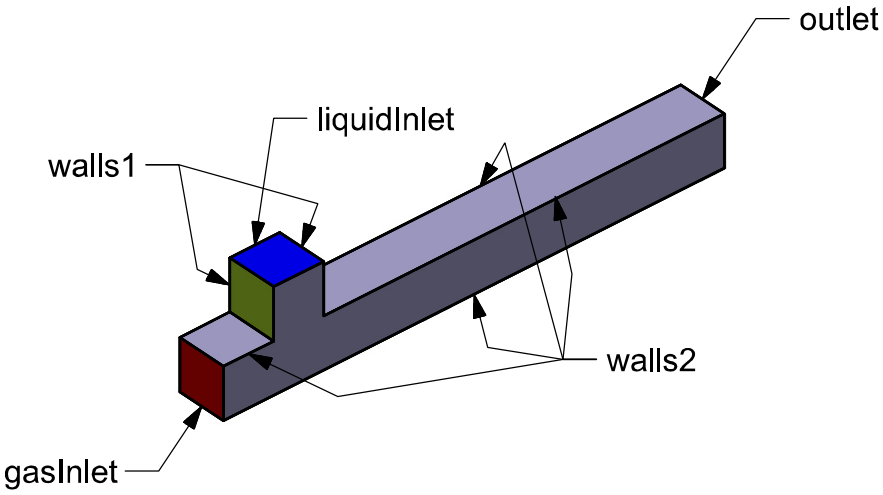


Figure 2.7: Patches scheme. Isometric and bottom view[4].

[4]3D mesh geometry created in*NX Siemens 10.0* modelling software.

2.3.2.2. Tools for mesh generation

The mesh generation is one of the longest processes of this project due to the software compatibility and utilities. The tools and programs used were the following:

- GMSH program.
- *BlockMesh* tool from OpenFOAM.
- *SnappyHexMesh* from OpenFoam.
- OnShape website for geometry exportation.
- ANSYS Mesh Generation extension. Students package software.

First, *blockMesh* generates simple geometry meshes through a text file called *blockMeshDict*. Initially, to test the tool, we managed to create a cylinder as a mesh, but it was a slow process.

Then GMSH software appeared, which was much easier to use when it came to generate a complex geometry with a progressive mesh density. Plus, the command *gmshToFoam* could be used to convert the mesh to an OpenFOAM compatible format before proceeding with the computations. The problem was that the cells of the cylindrical T-junction mesh were tetrahedral and different in size. It was complicated to modify these, which caused the Courant number to rise enough to stop the computations abruptly.

Then another shot was given to the *blockMesh* tool. After weeks invested in the cylinder T-junction mesh, it came to our realization that the *blockMesh* tool does not allow the junction between two complex geometries as cylinders. They were overlapped instead.

SnappyHexMesh tool from OpenFOAM implied the use of *blockMesh* as well, so it was more complex and even slower than the rest of the programs and tools. The biggest drawback of this tool was that it is usueful to generate environmental meshes outside the established geometry, but not contained in it like our case was.

Finally, ANSYS was the final software used for mesh generation despite the chosen geometry. A geometry of the mesh had to be designed with a modelling software and imported to the mesh generator, so we used OnShape website to do that. Then, ANSYS created the mesh inside the geometry with the Mesh generator package included in the student software installed in the local Windows PC. Adjusting the cell size and shape uniformly for the cylindrical T-junction mesh was not possible with the mesh generator package, so in the end, to avoid problems regarding the Courant number, the mesh geometry was simplified.

Using the command *"fluent3DMeshToFoam meshname.msh"*, the fluent mesh extension from ANSYS could become compatible with OpenFOAM and ParaView readers. The final geometry mesh had a $\Delta x = 4.375 \cdot 10^{-5} m$, which corresponds to an approximation of 290,000 elements, and a $\Delta t = 5 \cdot 10^{-6} s$.

2.3.3. Initial conditions

The initial conditions of the problem are defined in a folder named ”0” inside the simulation case. This name corresponds to the initial time, when the simulation starts. In it, since

temperature is constant along the process, the initial and boundary conditions are defined in three text files for velocity, pressure and air fraction. Inside those documents, the *internalField* is defined, which defines the initial state of the mesh for any property from above. The files regarding initial conditions are available in Appendix A.2.

In the velocity file, the internal field defines that the fluid of the mesh remains quiet until the simulation starts.

The pressure file states that the entire mesh is under atmospheric and free stream conditions initially, meaning that its gage pressure is *0 Pa*.

And finally, the air fraction file defines the value of $\alpha = 1$ for air and $\alpha = 0$ for water. Bubbles form because of the injection of air into the junction, so the internal field is fully defined as water ($\alpha = 0$) before that happens.

2.3.4. Boundary conditions

The boundary conditions are defined in the same files from the initial ones. The difference is that each boundary condition is related to a face group from the mesh. Section 2.3.2.1. shows that there are 5 differentiated face groups in this project's mesh, so there are at least 5 boundary conditions per file.

The boundary conditions might change depending on the interest behind each simulation, but the chosen ones are collected in tables 2.3, 2.4 and 2.5 for each field. Changing these do not affect the fluids' properties.

Path or wall	Condition type	Value (m/s)	Meaning
gasInlet	fixedValue	uniform (+0.1, 0, 0)	At any time, the injection of air to the tube is made at a fixed velocity of 0.1 m/s only in the positive x-axis direction.
liquidInlet	fixedValue	uniform (0, 0, -0.1)	At any time, the injection of water is made downwards the z-axis direction with a fixed velocity of 0.1 m/s.
outlet	zeroGradient	-	Applies a zero gradient condition from the internal field to the outlet patch, where no further conditions are imposed.
walls1 and walls2	fixedValue	uniform (0, 0, 0)	The fluid at the walls does not move in any direction. It can also be defined as a *"noSlip"* condition.

Table 2.3: Boundary conditions for the velocity field.

Path or wall	Condition type	Value (Pa)	Meaning
gasInlet and liquidInlet	zeroGradient	-	Applies a zero gradient condition from the internal field to both inlet patches, meaning that no further conditions are imposed there for pressure.
outlet	fixedValue	uniform 0	Applies a uniform value of 0 Pa in gage pressure to identify the outlet as a free stream outflow.
walls1 and walls2	zeroGradient	-	Applies a zero gradient condition from the internal field to the walls, where no other pressure conditions are imposed.

Table 2.4: Boundary conditions for the pressure field.

Path or wall	Condition type	Value (-)	Meaning
gasInlet	fixedValue	uniform 1	Indicates the entrance of air to the pipe through this inlet.
liquidInlet	fixedValue	uniform 0	Indicates the entrance of liquid to the pipe through this inlet, which is not different than the initial state condition.
outlet	zeroGradient	-	It does not impose any condition, as any mixture of air and water can be released as an outflow, depending on the bubbles.
walls1	fixedValue	uniform 0	Maintains the initial state condition at the wall, so liquid sticks to it.
walls2	fixedValue	uniform 0	Maintains the initial state condition at the wall, so liquid sticks to it. It can also have a contact angle condition.

Table 2.5: Boundary conditions for the air fraction field.

The conditions established generate a problem with the outflow and the bubbles do not exit the pipe normally. Some sort of vacuum is generated once the bubble reaches the outlet. Other conditions were tried (see Appendix C) but the scenario did not improve much. Therefore, these are the best conditions achieved and other projects have used these as well with no inconvenience so far ([15] and [18]).

Table 2.5, does not consider the contact angle condition. It is in the convergence tests phase that the conditions for walls2 change with a contact angle specification. This one says how the fluids stick to the wall or not. It is also known as wetting.

Wetting condition defines the angle between *walls2* face group in this case, and a liquid drop's boundary stuck in the wall. Usually the angle is defined from the inner part of the drop, as shown in the Figure 2.8. However, OpenFOAM defines the supplementary angle in its files. For example, for contact angle of 45^o, the file writes $\theta = 180^o - 45^o$.

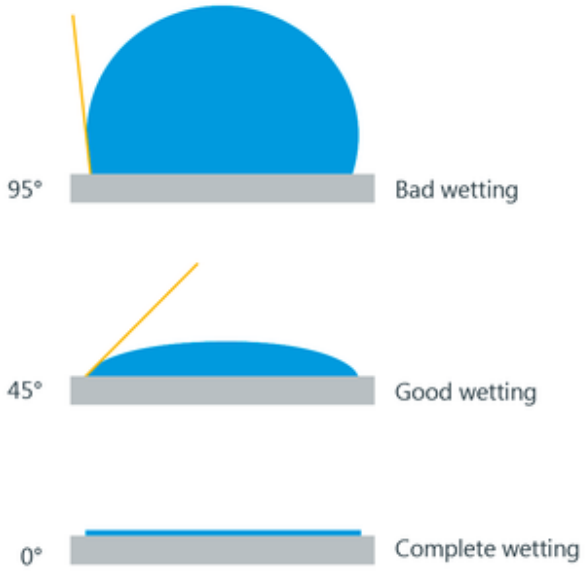


Figure 2.8: Contact angle and wetting schemes from inside the drop [19].

The files with the boundary conditions written for OpenFOAM are shown in Appendix A.2., including the contact angle condition.

2.3.5. Solver

Here, equations and warranty of the solver chosen in OpenFOAM will take place, considering the properties of the problem statement.

First of all, there are two fluids in multiphase at use in this project simulations. Therefore, a multiphase solver from OpenFOAM will be used for sure. Then, our mesh is static and does not present any optional motion nor change in its topology. In the mesh, neither air nor water change their phase along the tube. The interface between them is a mixture of both fluids, but it is small enough to consider them immiscible.

The previous statements narrow the list to two possible solvers:

- *InterFoam* solver, which works for two incompressible fluids, isothermal, immiscible fluids. This has interface capturing approach.
- *PotentialFreeSurfaceFoam* solver, which solves incompressible fluids through Navier Stokes. This enables a single-phase free-surface approximation.

Recalling the simulation properties from sections 2.2.1. and 2.2.2., air and water are considered incompressible and isothermal fluids. So far, the *PotentialFreeSurfaceFoam* solver does not specify any temperature condition. Also, both fluids run through the mesh and generate an interface between them that leads to free surface hypothetically. Air cannot create a free surface on its own and it needs water's help for that to happen. However, the problem is that water needs a gravitational field to form a free surface. So, considering a zero gravity condition, no free-surface can be created and the last solver does not apply for our case.

After all, the chosen solver is *interFoam*. This solver uses the VOF (Volume of Fluid) method, so that is why the air fraction property file needs to have at least one of the fluids defined at all times.

The mathematical process behind this solver is not seen by the user, but it can appears explained in OpenFOAM associated websites [20]. *InterFoam* solves the Navier Stokes equations of constant-density continuity and momentum (2.11 and 2.12) for both fluids. Their properties are constant when a cell is filled with only one of them and variable when the cell is located at the interphase. In the interphase, a weighted average of volume fraction computes the physical properties of each fluid, according to how much of the cell they occupy.

$$\frac{\partial u_j}{\partial x_j} = 0 \tag{2.11}$$

$$\frac{\partial(\rho u_i)}{\partial t} + \frac{\partial}{\partial x_j}(\rho u_j u_i) = -\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}(\tau_{ij} + \tau_{tij}) + \rho g_i + f_{\sigma i} \tag{2.12}$$

u = Velocity

g_i = Gravitational acceleration, neglected in this case

p = Pressure

τ_{ij} = Viscose stress

τ_{tij} = Turbulent stress

$f_{\sigma i}$ = Surface tension

ρ = Density

The density for the momentum equation 2.12 is the result of the following definition:

$$\rho = \alpha\rho_1 + (1-\alpha)\rho_2, \tag{2.13}$$

where α is 1 for ρ_1, and 0 for ρ_2. The density range between the values:

α = Air fraction value

ρ_1 = Density of the fluid 1 (AIR, ρ_G)

ρ_2 = Density of the fluid 2 (WATER, ρ_L)

The form of the previous equation is also used to find the general velocity vector (U) and viscosity (μ).

The surface tension ($f_{\sigma i}$) from the equation 2.12 is modelled as a continuum surface force (CSF). See the equation 2.14.

$$f_{\sigma i} = \sigma K \frac{\partial \alpha}{\partial x_i} \tag{2.14}$$

Equation 2.14 shows the dependency of $f_{\sigma i}$ on the x position of the cell, the air fraction (α) in it and the following two parameters.

σ = Surface tension constant, defined in files

K = Curvature

K is not a constant but a parameter, which can be found as:

$$K = -\frac{\partial n_i}{\partial x_i} = -\frac{\partial}{\partial x_i}\left(\frac{\partial \alpha / \partial x_i}{|\partial \alpha / \partial x_i|}\right) \tag{2.15}$$

After calculating the needed parameters in 2.13, 2.14 and 2.15; to solve the momentum and continuity equations (2.12 2.11), there is a final formula to determine whether there is an interphase between the two fluids or not in equation 2.16.

$$\frac{\partial \alpha}{\partial t} + \frac{\partial(\alpha u_j)}{\delta x_j} = 0 \tag{2.16}$$

Formula 2.16 requires an additional equation (2.17) to find the α parameter to determine the interphase. An advection function expresses the transport of this air fraction in time, using the general velocity vector found with an analogue equation of 2.13.

$$\frac{\partial \alpha}{\partial t} + \frac{\partial}{\partial x_i}(\alpha U) = 0 \tag{2.17}$$

2.4. Post-processing

This last section is about the post-processing of the collected data from the simulations. These include samples of a parameter or additional data manipulation to obtain certain results.

2.4.1. Brief methodology summary

Post-processing methodology involves the need to collect and organize data from simulations to present results. Then, sampled surfaces and probes from the mesh will be taken, which should be specified before the computations start in the cluster.

Matlab is the program in charge of the sampled data organization. Matlab data processing allows us to see the evolution of some of the computed bubble parameters to validate their values for a fully developed fluid in future convergence tests. It can also show the change of phase in a sampled surface throughout plots, as long as we write the code to do that.

2.4.2. Measurements

The measurements taken in simulations are made through sampled surfaces and probes. With these, the change of the air fraction in a surface or the pressure variation at some mesh nodes can be known.

The surface sampled values, points and probes need to be written in ”controlDict” file to specify OpenFOAM what it has to do with them. Taking samples does not add commands when running the simulation (see Appendix A).

In this project, there are several surface samples in order to analyse the evolution of the bubble parameters like its frequency, length, volume or speed as they run through the horizontal pipe. The data saved in a sampled surface relates to the alpha at each node of that surface. The value is written every 50 time steps[5]. The mean air fraction of each

[5]Remember that a time step is set to Δt = 0.000005 seconds yet.

surface at every time step is saved in separated files (see Appendix D).

If the bubble detachment takes place at the junction (between x = 1mm and x = 2mm), there will be 6 sampled surfaces per simulation, which are shown in Figure 2.9.

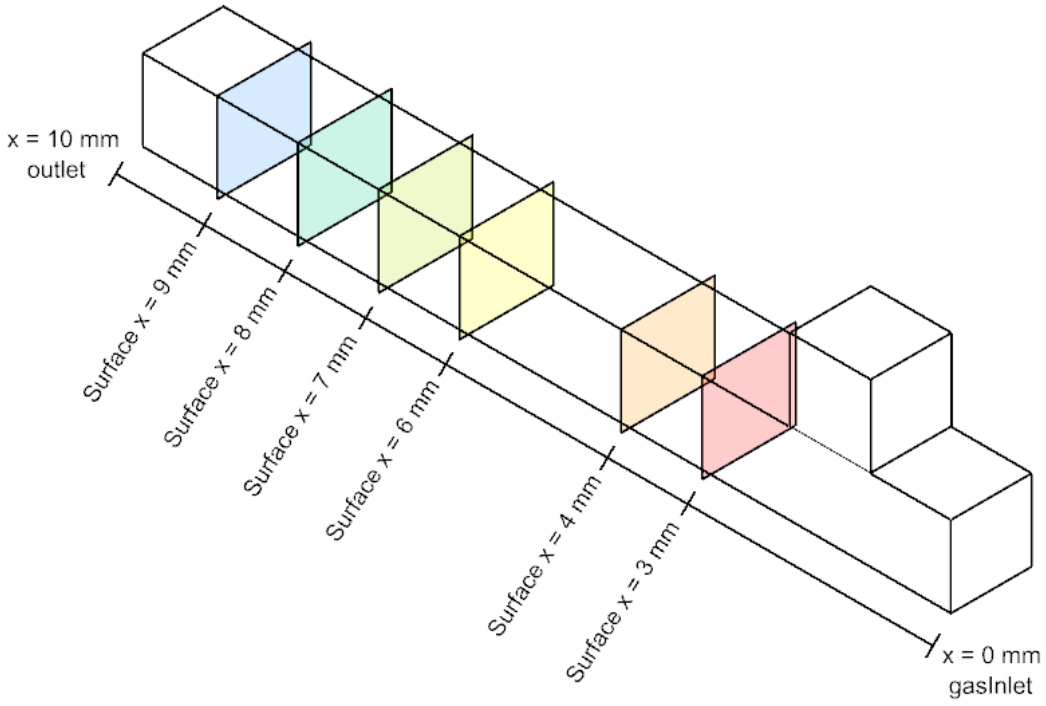


Figure 2.9: Sampled surfaces of the mesh[6].

Besides the surfaces of Figure 2.9, there is an extra sampled surface at y-axis midpoint to validate the pressure probes measurements with contour plots.

The probes save pressure values of 4 points of the mesh every time step. The biggest pressure variation must take place where the bubble detaches, so the points will be around the junction of the pipe, as shown in Figure 2.10. All points are in y = -0.0005 m.

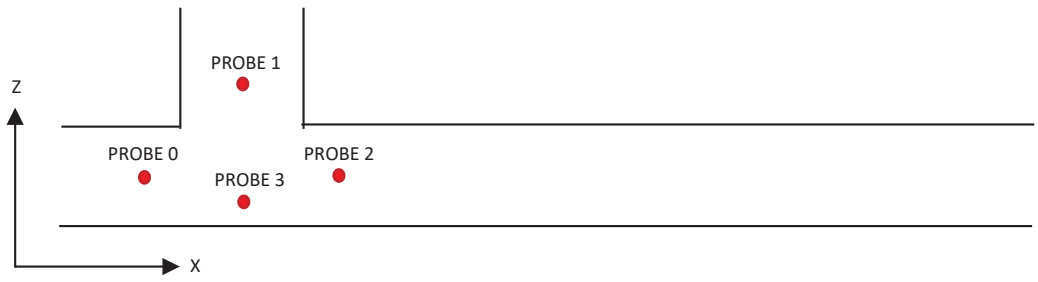


Figure 2.10: Probe points of the mesh for y-axis midpoint (y = -0.0005m).

[6]3D design created in *NX Siemens 10.0* modelling software.

The points from Figure 2.10 are chosen from the existent mesh nodes. Choosing any physical points lead to higher fluctuations if these are in the middle of a cell, between two or more nodes. Each probe can be identified according to their position, as written below.

- PROBE 0 = (0.000501092 -0.0005 0.000499999)m | *front point*
- PROBE 1 = (0.0015 -0.0005 0.00123913)m | *top point*
- PROBE 2 = (0.00250682 -0.0005 0.0005)m | *rear point*
- PROBE 3 = (0.00148303 -0.0005 0.00026087)m | *bottom point*

Further information regarding the probes and the Y-surface sampling file is available in Appendix D.

2.4.3. Matlab processing

The data collected in measurements is treated in Matlab to present the results and the convergence tests in future chapters.

2.4.3.1. Data management

The data collected in measurements is divided in: the air fraction data collection (independently of its writing interval) and the pressure probes at every time step. The purpose of the division is to manage them differently.

The pressure probes data is saved in a vector per column of the file (see Appendix D), and then plotted in Matlab.

The management of the air fraction values is more complicated though. In this case, there are two types of collected data: the mean air fraction along the time and all the real values per surface point. These are collected per each section.

The second type of collected data regarding the air fraction, saves the time of simulation and the sampled values in each surface point. With these, contour plots can be made in Matlab. An example of this is Figure 2.11, where the colour changes according to the value of α to differentiate air and water. The mean value of alpha at each section could not do that because it only collects one value per surface, regardless of its points.

No matter the complexity of the data measured in the surfaces, it is always stored in Matlab *structs*. The *structs* of the sampled surfaces own the main fields of simulation ”time” and ”data” to store all surface points and their values of air fraction. Matlab plots bubbles crossing a sampled section to see how many cross it and when they do, like in Figure 2.12.

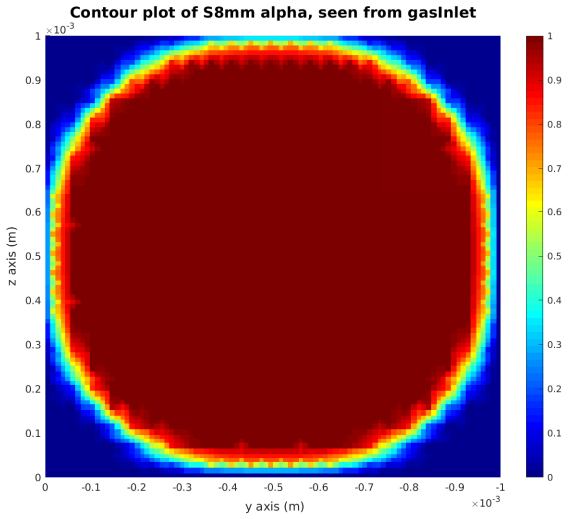


Figure 2.11: Example of contour at sampled surface 9mm when a bubble is crossing it.

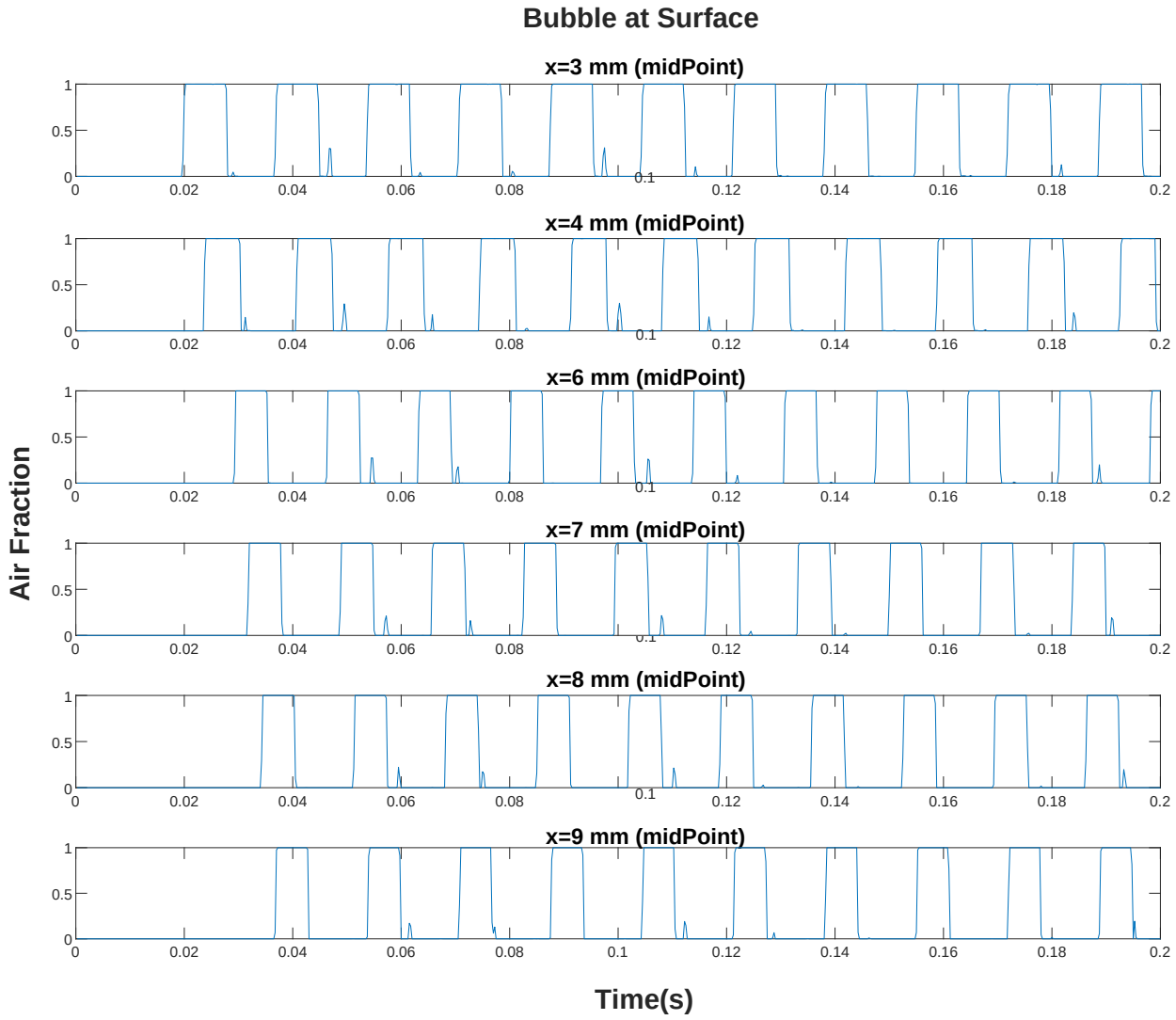


Figure 2.12: Air fraction at surface midpoint[7].

The *structs* from Matlab own two additional fields called ”meanAlpha” and ”realTime”. ”MeanAlpha” saves the mean air fraction value, and ”realTime”, the time of each mean value measurement. Then, Matlab plots these vectors to illustrate the physical behaviour of the bubbles and determine their real area. In Figure 2.13, the mean air fraction values never reach 1 like in the case of Figure 2.12 because of the average operation sampled. Several computations can be made from plots 2.12 and 2.13. The bubbles cross a surface

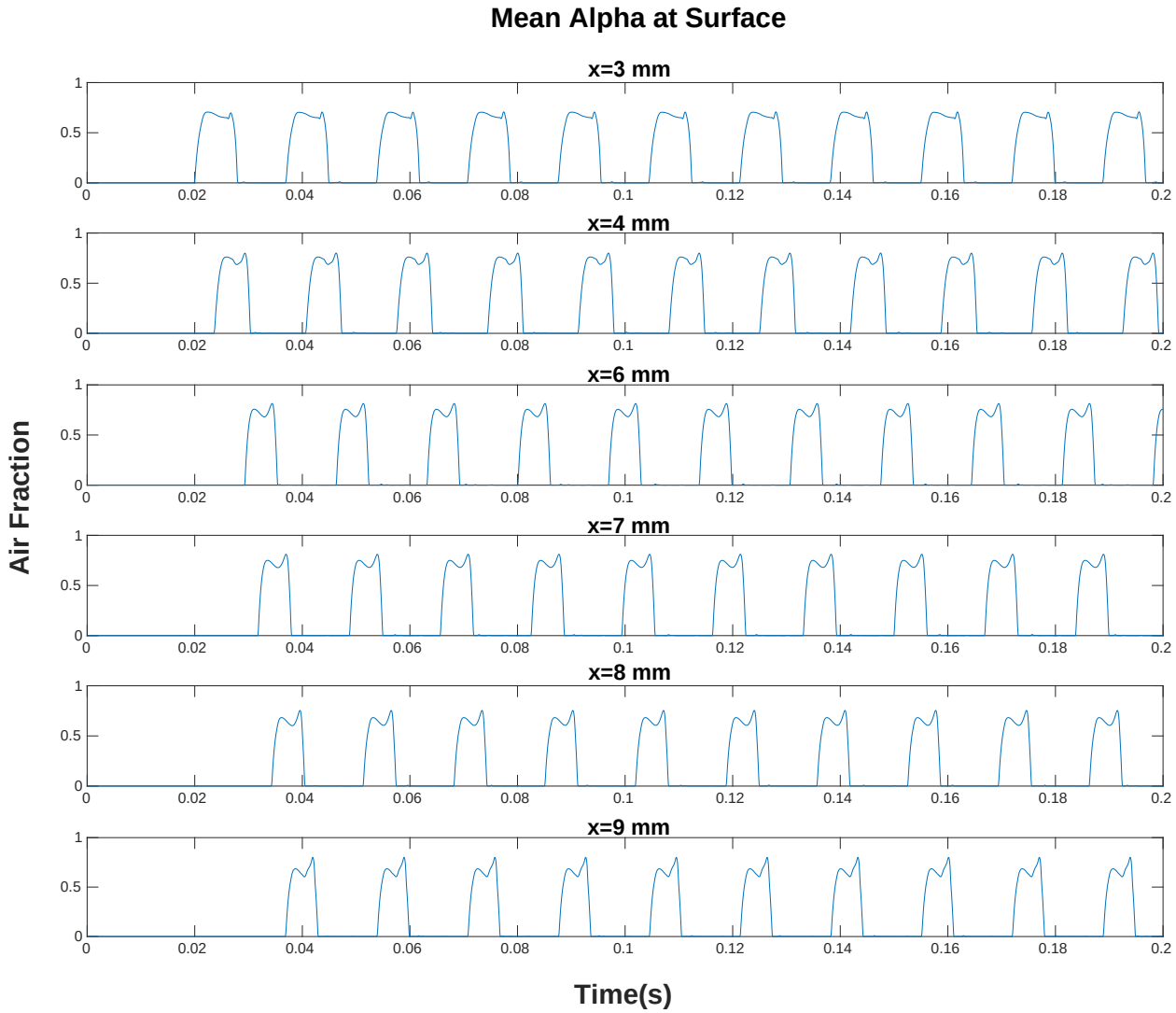


Figure 2.13: Mean air fraction evolution for each surface.

first through its midpoint and exits lasts from this point as well, so from plot 2.12, the initial and final times of a bubble are taken. However, many bubbles leave a trail behind, so from Figure 2.13, we filtrate the values of time according to the real bubble area. If a bubble is not big enough, this one is considered a trail and their initial and final points are discarded. The resulting points will be saved in a new *struct*.

The bubble parameters can be computed with the new *structs*, but first, the region that receives the most regular bubbles needs to be identified, as there the fluid is fully developed. This one will be called *capillary region*. Usually the fluid is fully developed at the closest region to the outlet, in this case that would be between sections x = 8 mm and x = 9 mm. Considering the odd outflow scenario though, it is better to take the previous region, the

one between surfaces x = 7 mm and x = 8 mm.
Matlab computation codes are available in Appendix E.

2.4.3.2. Parameters and computations

Four parameters can be computed (in S.I. units) from the *structs* of initial and final bubble times to define their behaviour:

- Bubbles' frequency (f_B).
- Bubbles' length (L_B).
- Bubbles' volume (V_B).
- Bubbles' velocity along the pipe (U_G).

Figure 2.12 selected times are useful to calculate the frequency, length and velocity of the bubble, while the area obtained from Figure 2.13 calculates their volume. The first set of bubbles usually provide strange parameter values though.

The frequency of the bubble is computed through the inverse of the consecutive bubbles period ($T_{B,8}$), as shown in Figure 2.14. This computation is made at section x = 8 mm.

$$f_{B,stable} = \frac{1}{T_{B,8}} \tag{2.18}$$

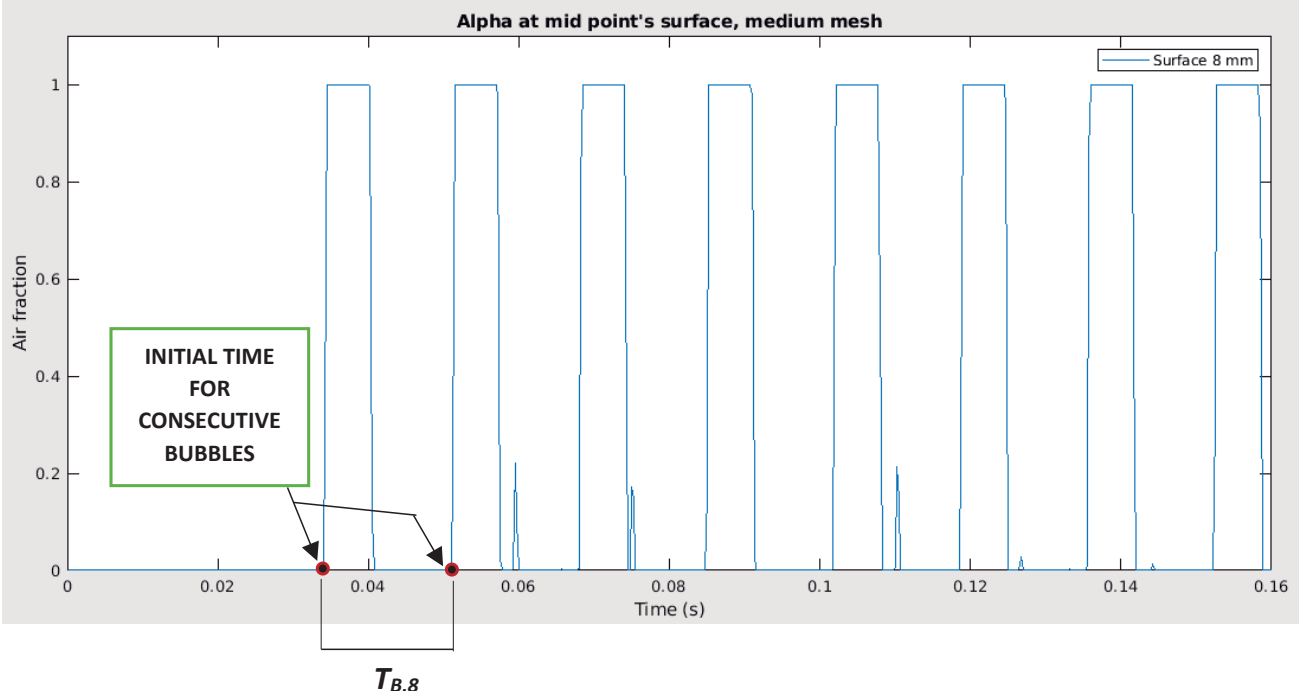


Figure 2.14: Period through the mean air fraction plot in Surface 8mm.

The length, area and volume of the bubble are also computed at x = 8 mm. Nevertheless, both length and volume depend on the stable speed ($U_{G,78}$), calculated from surfaces 7 and 8mm.

The length of the bubble also depends on the difference between the final and initial points of a bubble in Figure 2.15 from below.

$$L_{B,stable} = U_{G,78} \cdot (t_{final,i} - t_{initial,i}) \tag{2.19}$$

$U_{G,78}$ = Velocity of the bubbles when crossing the region 7-8 mm

$t_{final,i}$ = Exit time from the bubble "i" to surface 8 mm

$t_{initial,i}$ = Entry time from the bubble "i" to surface 8 mm

The equation 2.19 shows a different nomenclature for a bubble "i", where "i" is the number of bubble crossing the sampled surface at 8 mm. Examples of this nomenclature are shown in Figure 2.15.

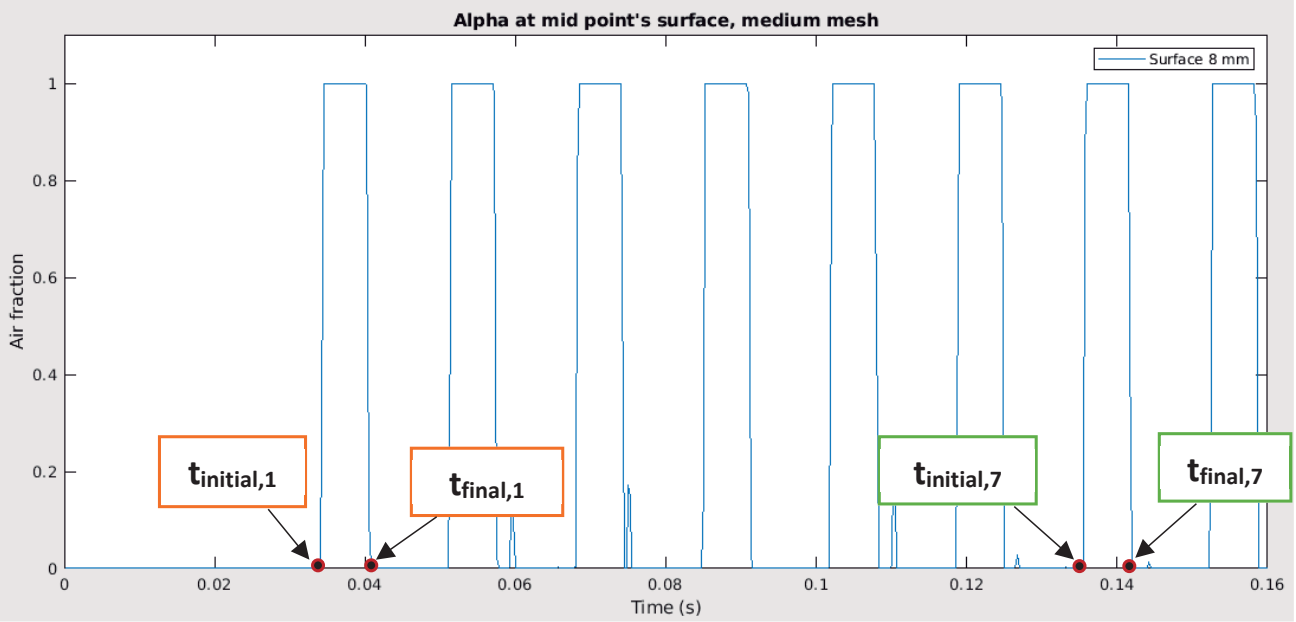


Figure 2.15: Entry and exit points of each bubble at surface 8mm.

The volume depends as well on bubbles real computed area from surface x = 8 mm, shown in Figure 2.16. The volume takes into account the squared section of the pipe, where each side of the section measures 1mm (see equation 2.20).

$$V_{B,stable} = A_{section} \cdot A_B \cdot U_{G,78} \tag{2.20}$$

$A_{section}$ = Area of the pipe's section (0.001^2 in this case)

A_B = Real area of the bubble

Figure 2.16 has a bubble painted in pink colour to show the area that Matlab computations will take into account.

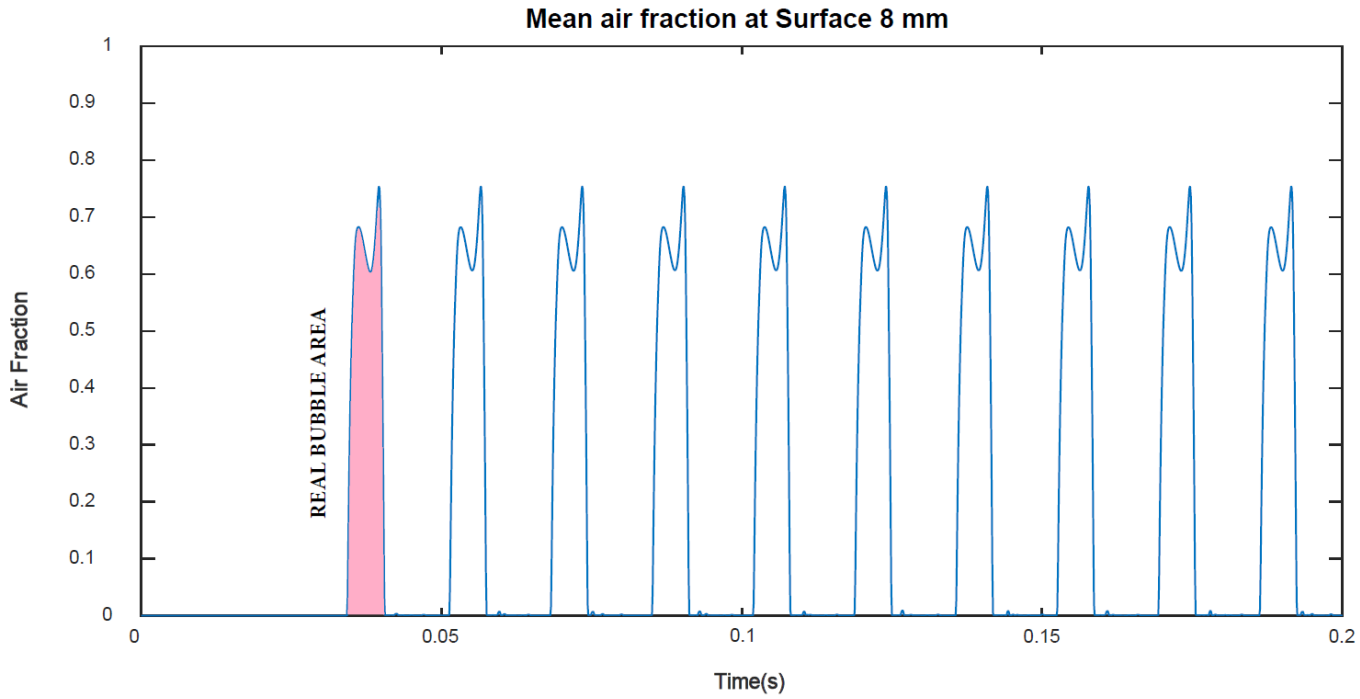


Figure 2.16: Real area bubbles crossing section at x = 8mm surface.

The bubbles' velocity is the only parameter that is computed considering two surfaces. When having 6 sampled surfaces along the x-axis (see Figure 2.9), four velocities can be computed from here: $U_{G,34}$, $U_{G,67}$, $U_{G,78}$ and $U_{G,89}$. The stable speed is $U_{G,78}$ (also called $U_{G,stable}$) and it appears in volume and length formulas (2.19 and 2.20). The velocity computation depends on the distance between two sampled surfaces and the initial times for a bubble crossing those two. This is shown clearly in the equation 2.21, which is applicable for any pair of sampled surfaces.

$$U_{G,stable} = U_{G,78} = \frac{X_2 - X_1}{t_{initial,2} - t_{initial,1}} \tag{2.21}$$

X_2 = Position of the second surface the bubble crosses(x = 8mm in case)

X_1 = Position of the first surface the bubble crosses(x = 7mm for stable speed)

$t_{initial,2}$ = Entry time of the bubble to the second surface

$t_{initial,1}$ = Entry time of the bubble to the first surface

2.4.3.3. Speed evolution

To determine if the computed parameters show a logical evolution, the evolution of the velocity along the T-junction will be evaluated.

Our simulations go from from t_0 = 0 s to t_{end} = 0.2 s. In that time, 10 bubbles are generated for a 290,000 elements mesh and a time step of 0.000 005 s.

Ideally, the bubble speed for the surfaces closest to the junction (x = 3 mm and x = 4 mm) should not be much higher than the sum of both inlet velocities (U_M). For $U_{SG} = U_{SL}$ = 0.1 m/s, U_M = 0.2 m/s.

According to the boundary conditions established, after crossing the first couple of sampled surfaces, a bubble's velocity should increase a little along the pipe until it reaches a stable value. Then, the velocity remains somehow constant. The stable value should never be greater than the double of the initial sum (U_M).

In Figure 2.17, it is possible to see the real behaviour of the 10^{th} bubble in the simulation.

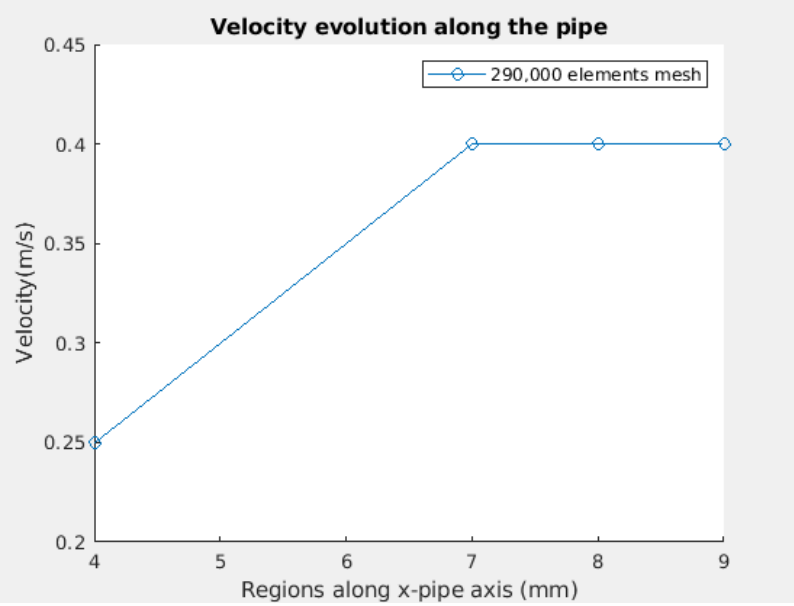


Figure 2.17: Velocity evolution along the sampled surfaces.

The velocity values computed are set in x = 4 mm, x = 7 mm, x = 8 mm and x = 9 mm in Figure 2.17 to indicate the end of each sampled region (3-4, 6-7, 7-8 and 8-9).

Figure 2.17 shows that the bubble velocity sticks to the predicted behaviour written above. The velocity value never exceeds $0.4m/s$, which is the double of the inlet velocities sum, and the value between sections 3 and 4 is $0.25m/s$. All the values adjust to what we wanted then. This also proves that the stable speed belongs to the constant region and that this value is indeed a good choice to represent the fully developed fluid. It is true that regions 6-7mm and 8-9mm have the same value, but these cannot be chosen for two reasons.

1. Region 6-7 conditions is between two states of the bubble, where it goes from increasing its speed to stabilize on a certain value. That makes the region transient.

2. Region 8-9 is discarded by its closeness to the outlet and the rareness of the outflow. It may give unexpected parameters.

In the end, the chosen capillary region where the flux is fully developed does not present variations and it is far enough from the outlet to consider its behaviour "average". Considering this one the expected behaviour, the mesh seems to be stable enough to proceed with the convergence tests from chapter 3.

CHAPTER 3. CONVERGENCE TESTS

This chapter compares meshes of different cell size, time steps and contact angles to obtain a higher accuracy in results' chapters (4).

3.1. Mesh convergence

The mesh convergence tests compare the bubble parameters between three meshes with different cell sizes and elements. Their overall time step considered in this test is $5 \cdot 10^{-6}$ seconds. The convergence test is made to the following meshes.

- Coarse mesh of about 145k elements. It lasts less than 12 hours in the cluster.
- Medium mesh of approximately 290k cells. It lasts a day in the cluster and this is the mesh used until this section.
- And a fine mesh of about 435k elements. This one lasts more than two days computing in the cluster.

The parameter comparison of this test will be made at the capillary region, where the fluid is fully developed. A plot of the bubbles crossing surfaces 7 and 8, the variation of parameters per bubble at the surface x = 8 m and a table collecting the computed parameters per mesh will be shown.

Figure 3.1 illustrates the first difference in bubbles according to the mesh. For the same period of time, the coarse mesh creates 10 thin bubbles while the others make less. This proves that the coarse mesh has a higher "bubble velocity", caused by the big cell size and the lack of accuracy it causes to the computations.

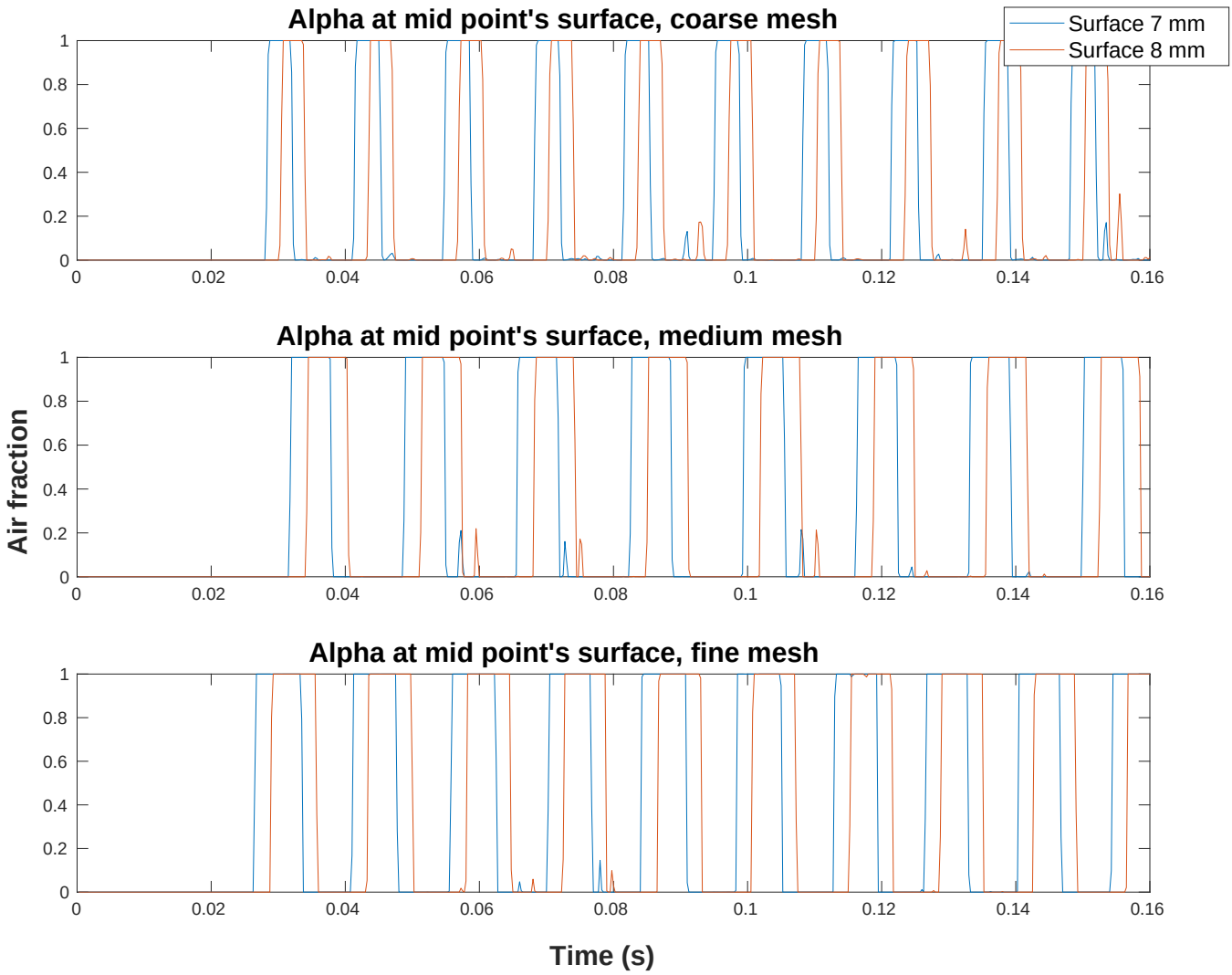


Figure 3.1: Bubbles from each mesh crossing surfaces x = 7mm and x = 8mm.

Each mesh has associated the parameters of frequency, length, volume and stable velocity of the bubble in this test. The bubble volume dependencies (2.4.3.2.) makes the comparison between bubble velocities be enough to determine its difference in each mesh. Therefore, Figure 3.2 illustrates the variation of frequency, length and velocity parameters per bubble crossing the surface with the most developed flux. The plot includes the first 9 bubbles released per mesh, as one of them only releases that amount. The medium mesh has the lowest bubble velocity, while the others differ greatly from it. We believe that this unexpected outcome might come from the rare resulting outflow.

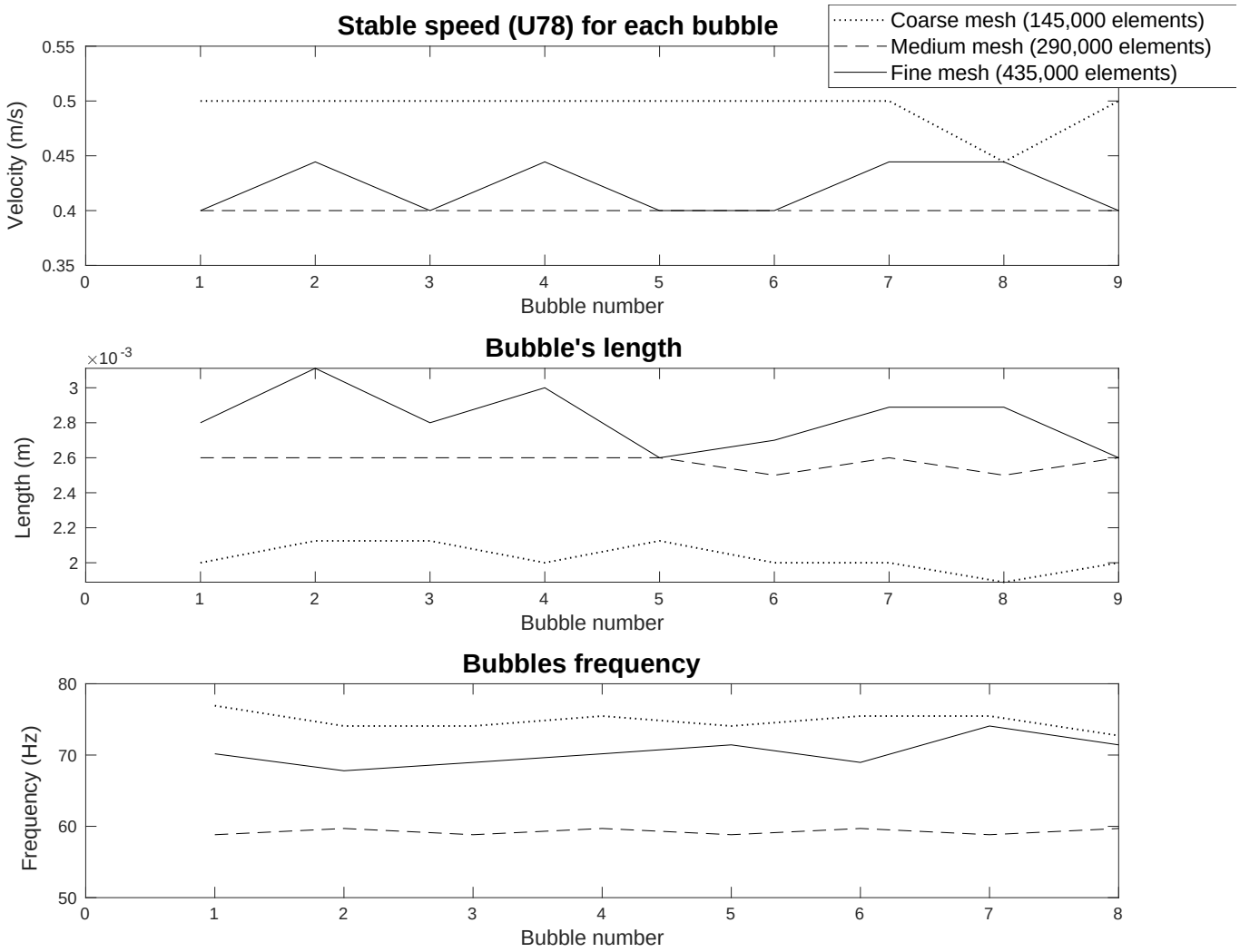


Figure 3.2: Parameters evolution per bubble and mesh.

Table 3.1 shows the parameters computed for the 9^{th} bubble of each simulation. The finer the mesh is closest to the correct results, so the errors are calculated respect to the finest mesh values.

Mesh	$U_{G,78}$ (m/s)	ε_U (%)	L_B (mm)	ε_L (%)	f_B (Hz)	ε_f (%)
145k cells	0.50	25.0	2.00	23.1	72.7	1.8
290k cells	0.40	0.0	2.60	0.0	59.7	16.4
435k cells	0.40	-	2.60	-	71.4	-

Table 3.1: Mesh parameters comparison.

The 145k cells mesh errors are smaller for the frequency value, where its behaviour is quite similar to the finest mesh, according to Figure 3.2. That mesh resemblance should not happen in the fully developed flux sampled surface, since both meshes present a lot of variations that show instability. Despite the frequency error, the 290k elements mesh seems to be the most stable mesh out of all three. The boundary conditions at the outlet

may manipulate the outcome of table 3.1, but the medium mesh parameters have a better physical behaviour in the flux developed surface.

The timing of computations in the cluster is quite long for the fine mesh in comparison with the other two meshes. Seeing the variations and similarities from Figure 3.2 and Table 3.1, the extra time taken for computations is not worth it in this case. It is also known that coarser meshes does not provide the best accuracy and presents oscillations in values. These statements confirm that the 290k cells mesh is the best fit for future simulations due to its stability and ”short” computational time (24 hours). The errors committed in it respect to the finest for frequency, length and stable velocity of the bubble are 16.4%, 0% and 0%, respectively.

3.2. Time convergence

The time convergence uses the 290k cells mesh and tests computations time steps that do not jeopardize the Courant number. Ideally, the smaller the time step is, the closer the results are to convergence. Then the time steps to study will be the following:

- Big time step of $1 \cdot 10^{-5}$ seconds. It lasts half-day in the cluster computing.
- Medium time step of $5 \cdot 10^{-6}$ seconds. Its the current one and it lasts about a day calculating.
- Small time step of $2.5 \cdot 10^{-6}$ seconds. It lasts about one and a half days in the cluster.

Now that the mesh is the same in all tests, it is assumed that there should not be such big difference between the three simulations.

The shortest number of bubbles generated in the 3 simulations for this test is 9. So the parameter comparison will be made for the first 9 bubbles.

As it happened in the mesh convergence test, the bigger time step generates more bubbles than the rest cases for a same period of time. When the biggest time step simulation detaches 9 bubbles, the other two have not made the half of it. Figure 3.3 illustrates this for the capillary region of fully developed flux.

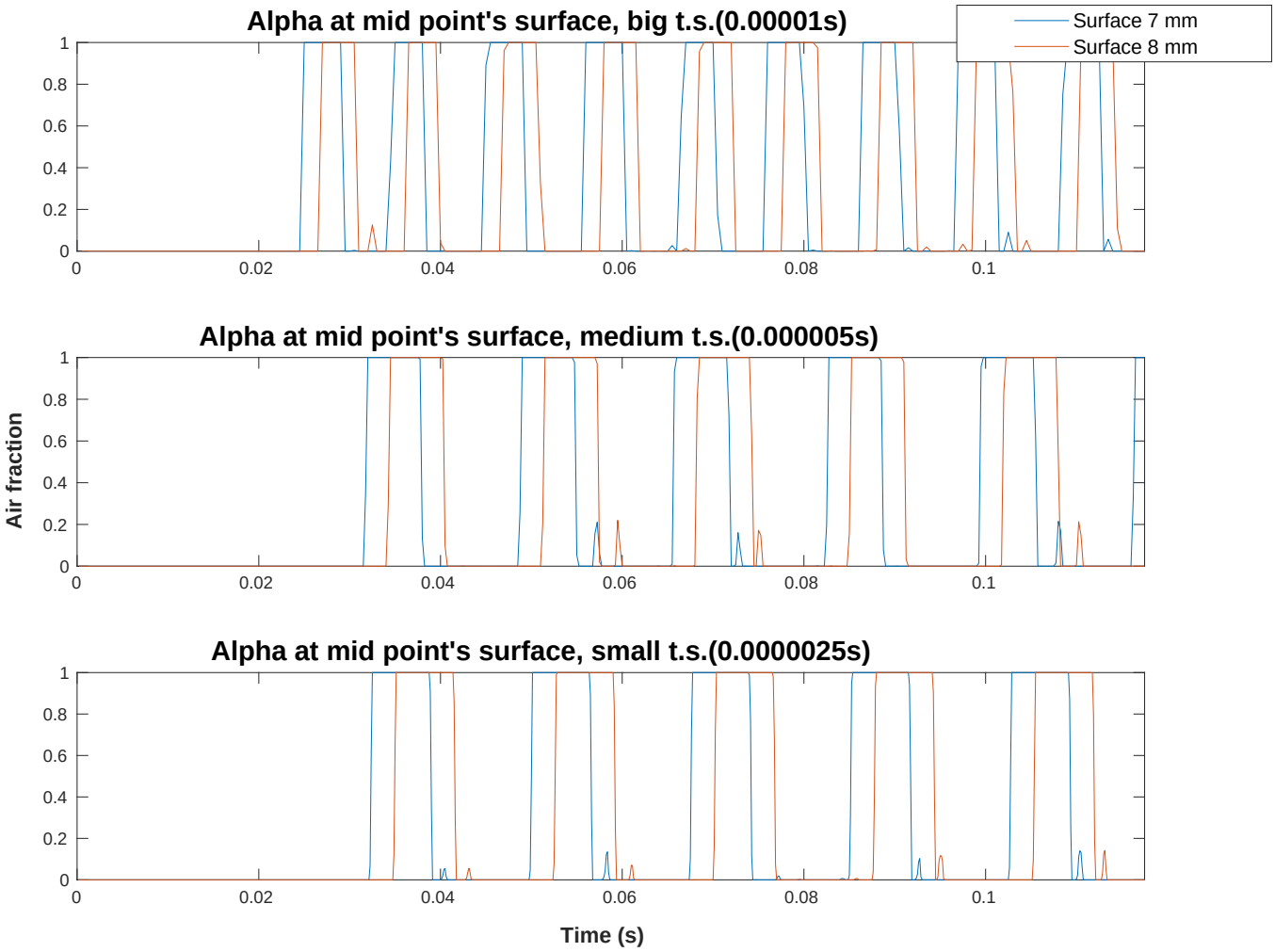


Figure 3.3: Bubbles crossing surfaces 7 and 8mm for each time step simulation.

The medium and small time steps simulations make the same number of bubbles as it is seen in Figure 3.3, which demonstrate that they are both closer to convergence. The biggest time step simulation should not present the best results and be out of the picture in no time if Figure 3.4 and table 3.2 prove us right.

Figure 3.4 presents the parameters variation for the first 9 bubbles to see if their behaviour is stable as expected or not.

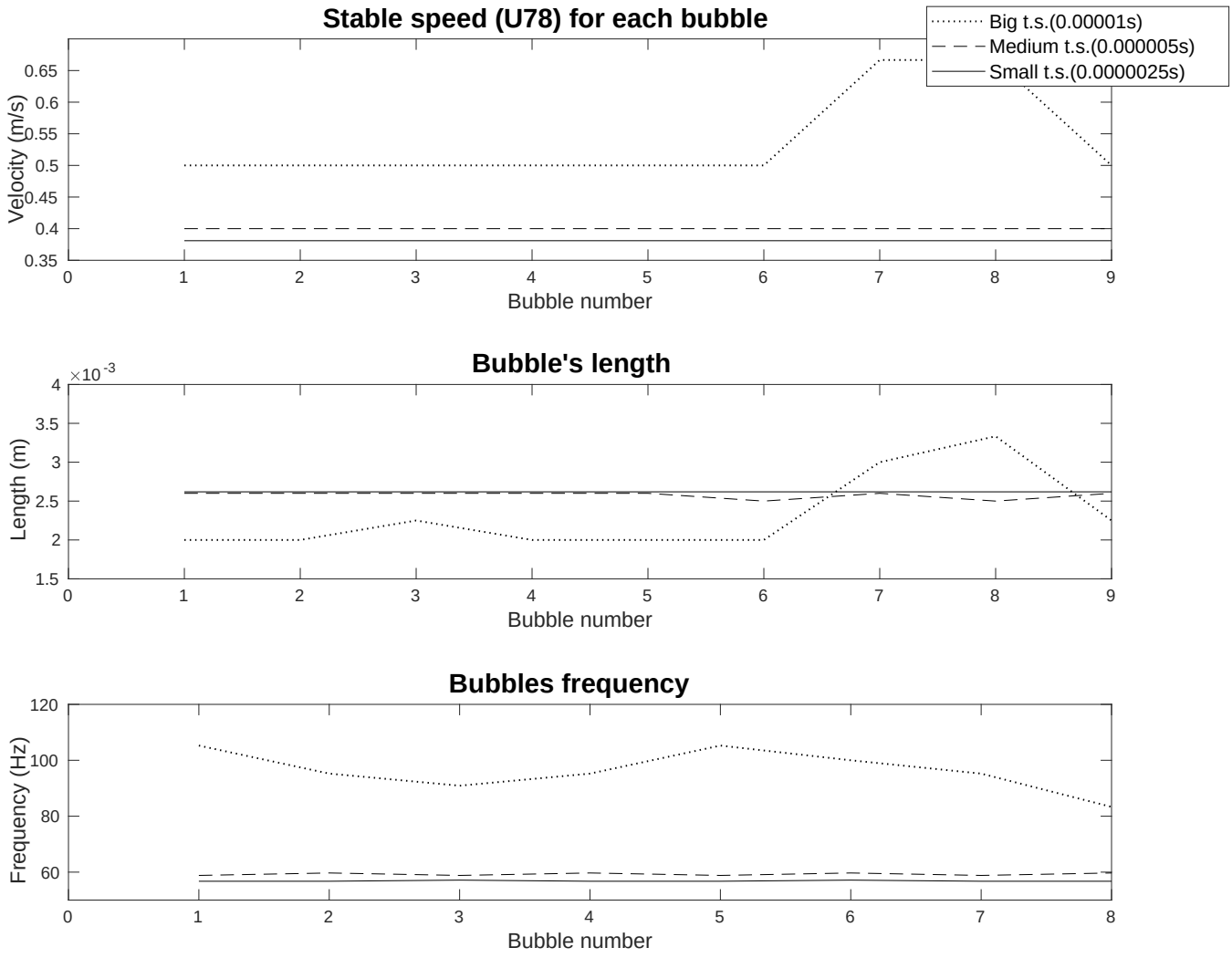


Figure 3.4: Parameters evolution per bubble and time step.

Contrary to the mesh convergence results, Figure 3.4 plots present a reasonable physical behaviour: the medium time step sticks the most to the smallest one in all parameters, meaning that they are both closer to convergence than the big time step computations. Also, the bigger time change values per bubble while the other two present practically a straight line. Apparently, the stability of the mesh used brings parameters to stable conditions when they are closer to convergence.

Being the medium and small time steps results so alike implies that the extra hours for the smallest time computations are not worth it. Table 3.2 must confirm the veracity of this statement. The errors presented are respect to the smallest time step, whose results are considered the most convergent in this case.

Time step	$U_{G,78}$ (m/s)	ε_U (%)	L_B (mm)	ε_L (%)	f_B (Hz)	ε_f (%)
$1 \cdot 10^{-5}$ **seconds**	0.50	31.3	2.25	14.1	95.2	46.9
$5 \cdot 10^{-6}$ **seconds**	0.40	5.0	2.60	0.7	59.7	5.2
$2.5 \cdot 10^{-6}$ **seconds**	0.38	-	2.62	-	56.7	-

Table 3.2: Time convergence parameters comparison.

Table 3.2 confirms the predicted statement. The errors between the medium and the short time steps are small enough to consider the medium time step results good and close to convergence. Therefore, the time step $5 \cdot 10^{-6}$s is chosen with errors in frequency, length and speed of the bubble of 5.2%, 0.7% and 5%, respectively.

3.3. Contact angle simulations

With the 290k mesh and $5 \cdot 10^{-6}$s time step, it is time to test different contact angle values to improve the performance of our simulations. According to the conclusions of Arias and Montlaur (2017) [1], the chosen contact angle value was 25°for the walls of the horizontal pipe. We can expect a similar value to work but several contact angles will be tested just to be sure.

The tested angles vary from 0°to 90°to make the fluids stick to the wall when running through the horizontal pipe. Within the selected range, smaller angles increase the water adhesion to the wall, while the largest do the opposite. Angles greater than 90°do not guarantee the adhesion of any fluid to the wall.

When no contact angle is determined, any fluid can stick to the wall depending on the bubbles' trajectory through the pipe. Figures 3.5 and 3.6 illustrate that behaviour. They show captions and contour plots of a bubble in each simulation crossing surface x = 8 mm.

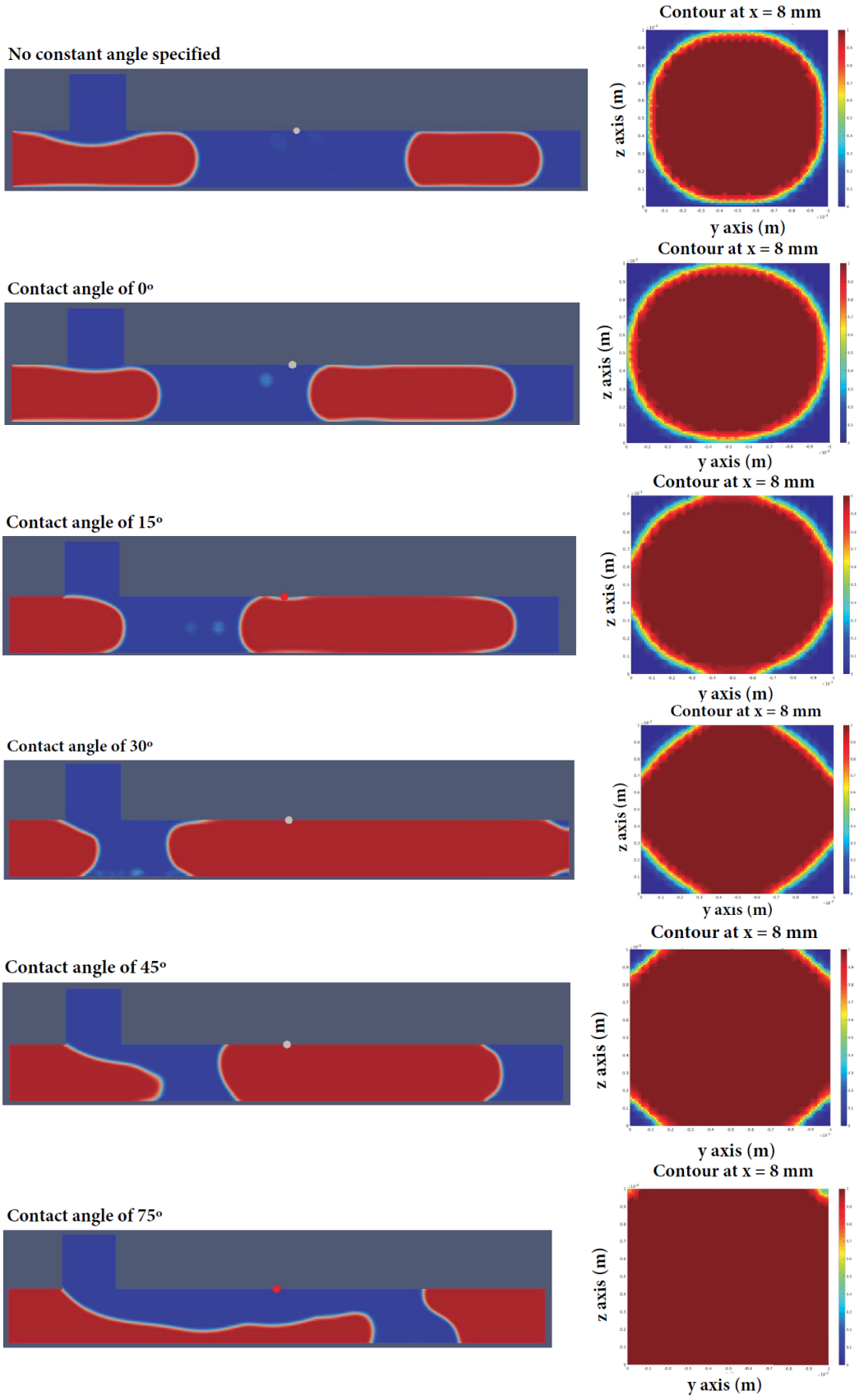


Figure 3.5: 8^{th} bubble captions for no angle, 0°, 15°, 30°, 45°, 75°conditions[1].

[1]Pictures from the left is a caption of ParaView, the one on the right is the contour plotted in Matlab.

Figure 3.6: Caption and contour of the 2^{nd} bubble crossing surface 8mm for 90°[2].

When the gas adhesion to the wall is too strong, the number of bubbles generated decrease. That is why the captions in 3.5 show all the 8^{th} bubble, because the 75°cannot produce more. The angle of 90 degrees is exempt because it can only generate 2 bubbles. It is known that the first bubbles are always unstable and do not describe the real physical behaviour of the simulation. Actually, Figure 3.6 presents the second and last bubble generated by a 90°angle, and the final scenario it leaves after releasing the second bubble looks like Figure 3.7.

Figure 3.7: Caption and contour of the 90°after the 2^{nd} bubble is released[3].

Bubbles' length increases with the contact angle until overcoming 45°. Then, the gas sticks to the wall so much that the bubbles become shorter as they occupy more area of the pipe's section. All contact angles have a larger bubble than the non-specified angle simulation, though. Therefore, neither the non-specified angle nor the angles above 45 degrees will be chosen as they do not lead to realistic results. Plus, the largest angles generate less bubbles and that adds difficulties to the study. Therefore, as expected from [1], the most promising simulations have contact angles of 0°, 15°, 30°and 45°.

Matlab plots (figures 3.8 and 4.1) show the values of the length and the velocity of the 8^{th} bubble for no contact angle, 0°, 15°, 30°, 45°and 75°. These values take as a reference the no-contact-angle-specified case just because of its regular behaviour. The chosen contact angle should not be present values that differ too much from that case.

[2]Pictures from the left is a caption of ParaView, the one on the right is the contour plotted in Matlab.
[3]Caption extracted from ParaView.

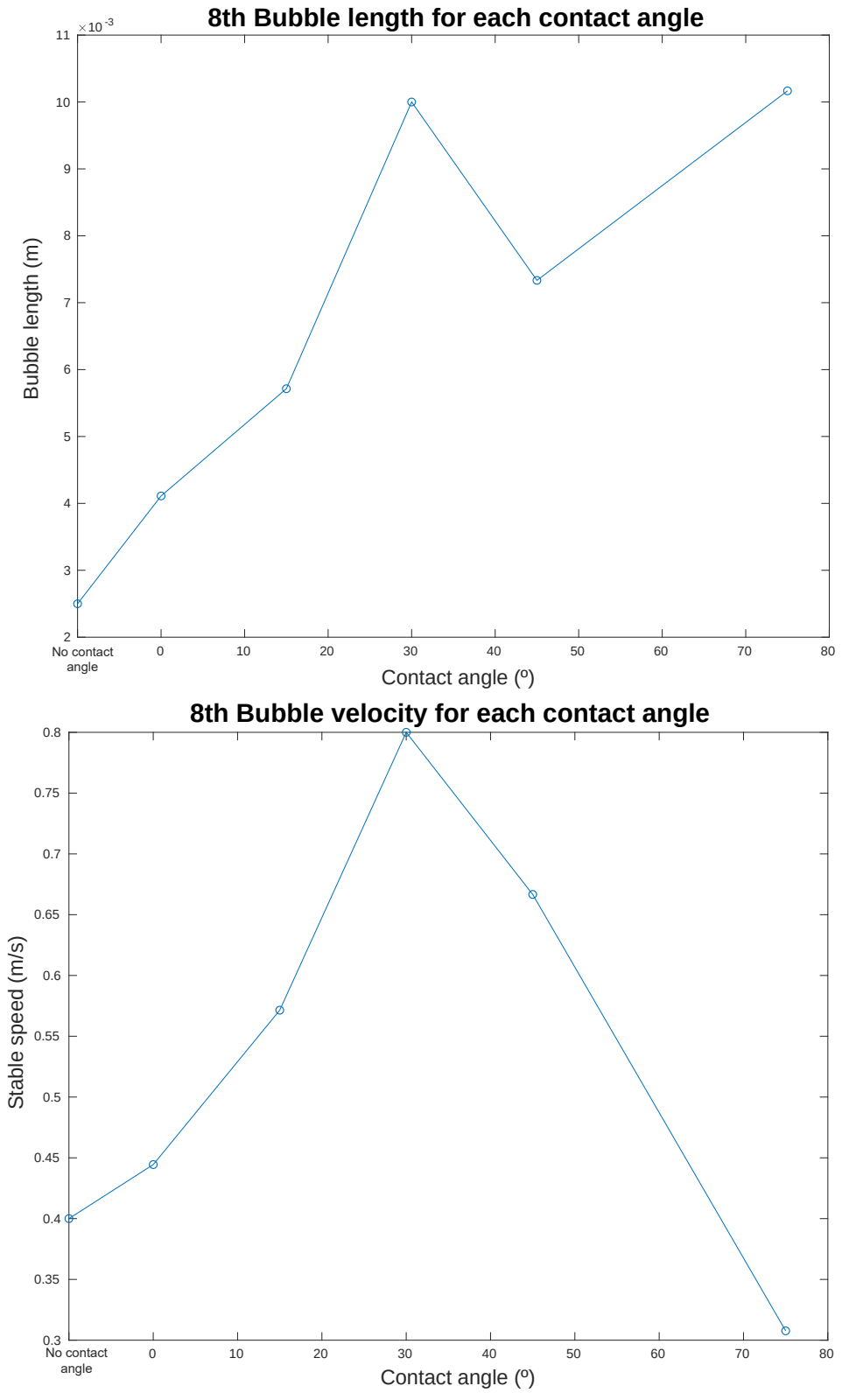


Figure 3.8: Length(up) and velocity(down) of the bubble as a function of the contact angle.

The velocity plot shows much higher velocities for the angles of 30°and 45°than they should. This leads to strange values in length as well, since it depends directly from stable speed. Our guess is that the vacuum effect of the outflow is stronger for these values, so they will not be of use. Then, from the angles left, it is better to pick the 15°because it is

closer to the value taken in [1] and to the real physical result. Otherwise, there would be no difference between the 0°and the non-specified angle case.

The tests performed in this chapter have lead us to one conclusion. The simulation that brings most stability to the simulations has a 290k cells mesh, uses a time step of $5 \cdot 10^{-6}$ seconds for computations and a contact angle of 15 degrees. This one lets the water still stick to the wall and accelerate the bubbles through the horizontal pipe with the intended behaviour.

The results chapter (4) will take these features in their simulations.

CHAPTER 4. RESULTS

With a 290k cells mesh, the time step of $5 \cdot 10^{-6}$ seconds in computations and the contact angle of 15°, it is time to see what results can be obtained from these conditions.

This chapter has two sections. First there is the analysis of pressure evolution at probe points during a bubble detachment, and then, we will try to evaluate the simulation with new inlet velocities for each fluid: U_{SL} and U_{SG}.

4.1. Pressure probes results

First let's study the pressure evolution when creating a bubble. This is made through the probes from section 2.4.2.. The data taken from them is processed in Matlab to present their pressure evolution plot. Matching contour plots of the Y mid-surface are also made to verify the process.

Figure 4.1 shows the pressure evolution along the 0.2 seconds of simulation time. The graph does not include the first instant though, so that pressure peaks can be avoided and the rest of the process can be appreciated. Injecting air for the first time introduces a huge amount of pressure to the initially stable system with 0 Pa of gage pressure.

Generally, Figure 4.1 presents similar behaviours between all the probes.

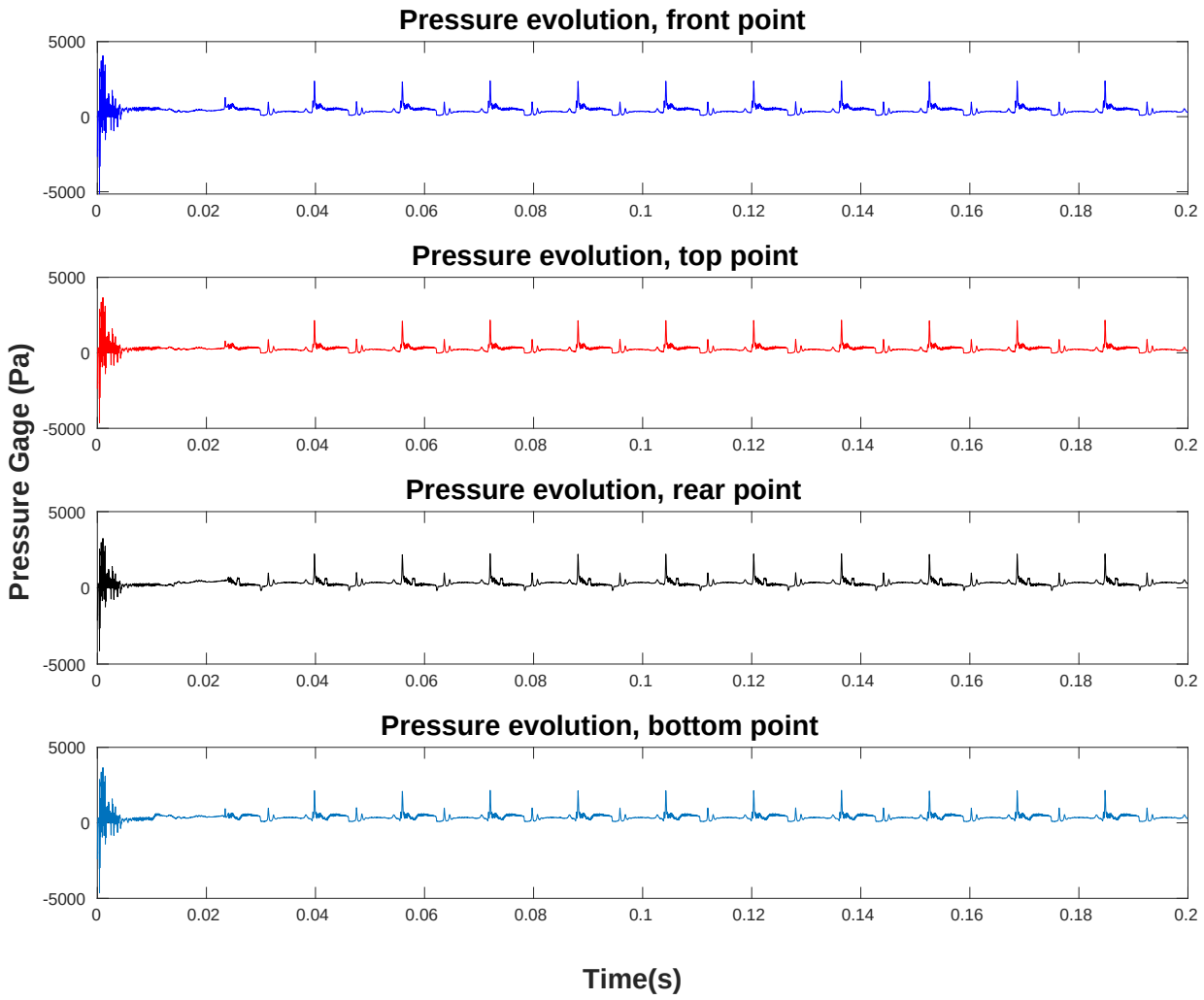


Figure 4.1: General pressure evolution for medium mesh, time step and 15°angle.

The pressure field is seen to be quite stable throughout the entire simulation, despite the variations in time every now and then. When a bubble detaches, the values of pressure are higher and fluctuate, although this is hard to notice in comparison with the first bubble produced.

Figures 4.2 and 4.3 show the zoomed region of the plot correspondent to the 8^{th} bubble of the simulation, so that the variations can be appreciated. The first image show the bubble detachment measures from the four probes separately, while the second one joins their evolutions to identify their differences.

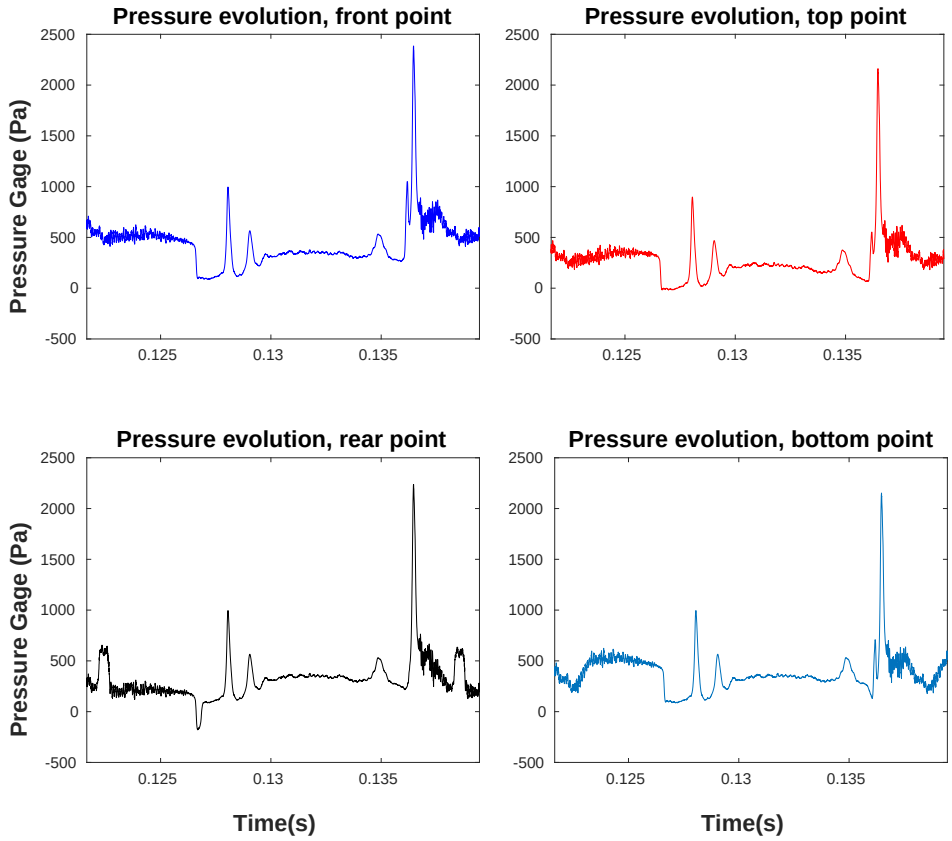


Figure 4.2: Pressure evolution of the 8^{th} bubble generation and detachment.

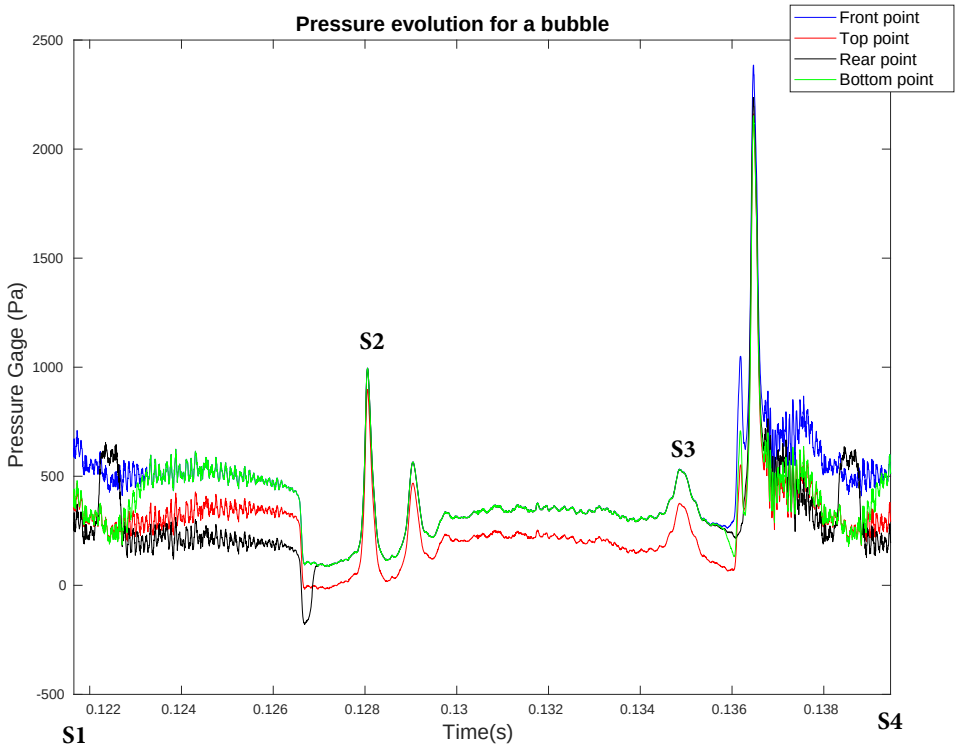


Figure 4.3: 8^{th} bubble superposed pressure evolution probes.

The time vector from Figures 4.2 and 4.3 cover from the moment the 7^{th} bubble leaves the four probes region (0.121 655 s), to the moment the 8^{th} bubble does the same (0.139 455 s). The process of the 8^{th} bubble detachment is summarized in captions from 4.4.

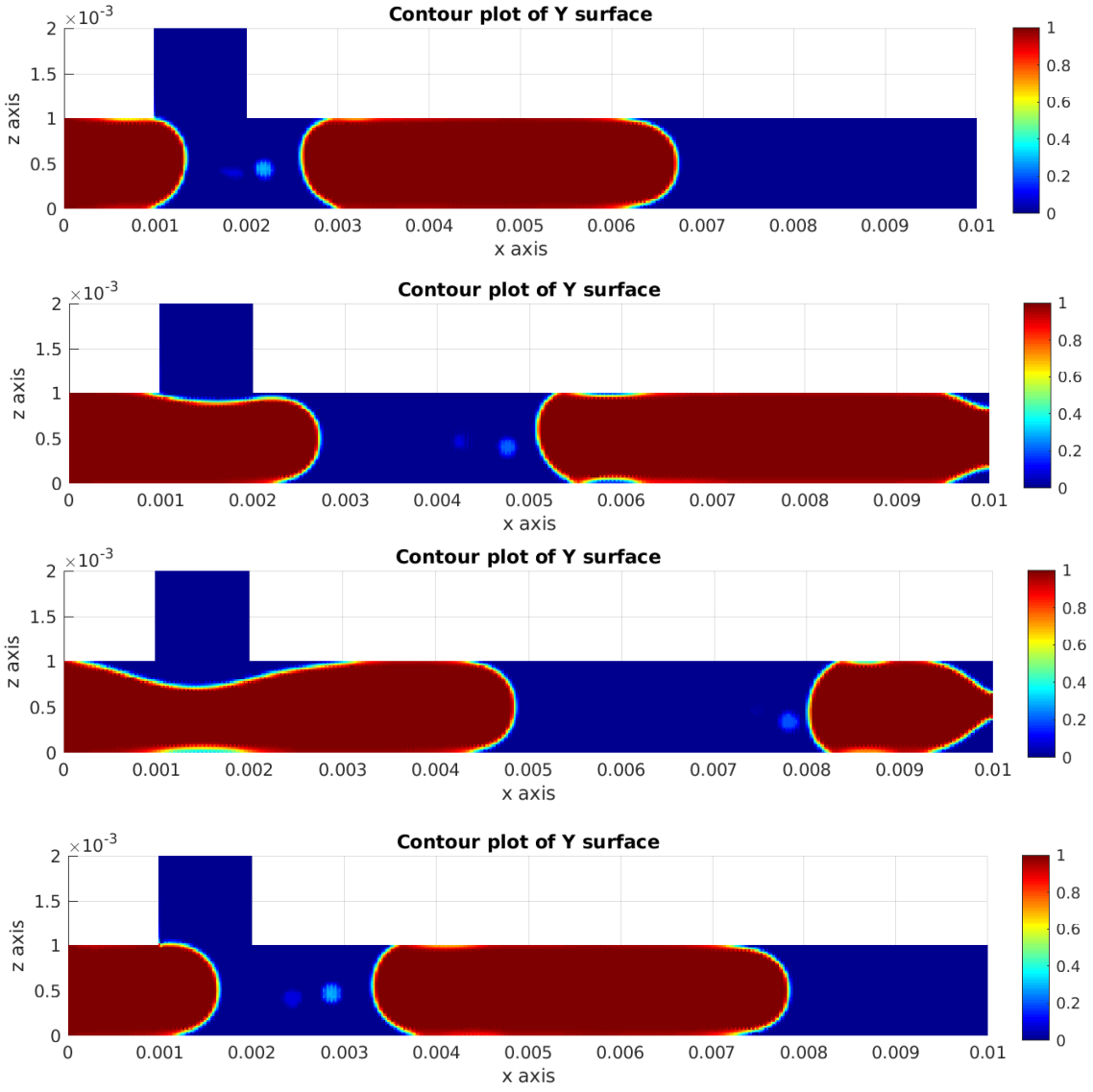


Figure 4.4: 8^{th} bubble generation sequence (S1, S2, S3 and S4 from 4.3).

From the previous images regarding the 8^{th} bubble, it is noticeable that the front point (probe0) is always in the gas region, so its pressure is slightly higher than the rest of the points. This is actually seen in Figure 4.4, where the overall values are higher, especially in early stages and the highest peaks of variation.

The top point (probe1) is always in the liquid region. The initial pressure is smaller until the gas covers the entire junction (Figure 4.4) and rises the pressure, creating the peaks in the red plot of Figure 4.2.

The rear point (probe2) has the lowest pressure at the early stages of Figure 4.3. It has a little variation caused by the trail the 7^{th} bubble leaves behind (Figure 4.4). Then just before the bubble detaches, the pressure rise of this point is the second highest of all. The

bottom point (probe3) presents quite average values and no other significant fluctuations.

All plots present the biggest change in pressure when the bubble starts detaching, around the time shown between the 3rd and 4th caption from sequence 4.4. That is the highest peak of pressure for all points, probably due to their closeness to one another.

4.2. Inlet speeds variations

Now it is time to try several inlet velocities (U_{SL} and U_{SG}) and see what happens with the frequency, length, velocity and volume of the bubble. Numbers in table 4.1 correspond to combinations, from case 1 to 9. The already tested inlet velocities are included in Combination 5.

U_{SL} \ U_{SG}	0.05 m/s	0.1 m/s	0.5 m/s
0.05 m/s	1	2	3
0.1 m/s	4	5 *(from previous chapters)*	6
0.5 m/s	7	8	9

Table 4.1: Velocity inlet combinations for further parameters discussion.

Initially, the tested values for both inlet velocities were supposed to range between 0.1 and 1 m/s. However, some combination between these two extremes could not detach a bubble because $0.01\,\mathrm{m/s}$ was too slow against $1\,\mathrm{m/s}$. Therefore, the range is adjusted to the values $0.05\,\mathrm{m/s}$ and $0.5\,\mathrm{m/s}$ to detach bubbles successfully.

The 7^{th} bubble will be evaluated in the capillary region where the flux is developed, since the slowest combination (number 1 in 4.1) only generates 7 bubbles. All the captions and values below are related to this bubble.

Figure 4.5 show the same bubble detached for all combinations, in order from top to bottom. The sequence is extracted from ParaView visualizer for most of the cases, except for the 5^{th}, which is a contour built in Matlab (see also the first caption of Figure 4.4).

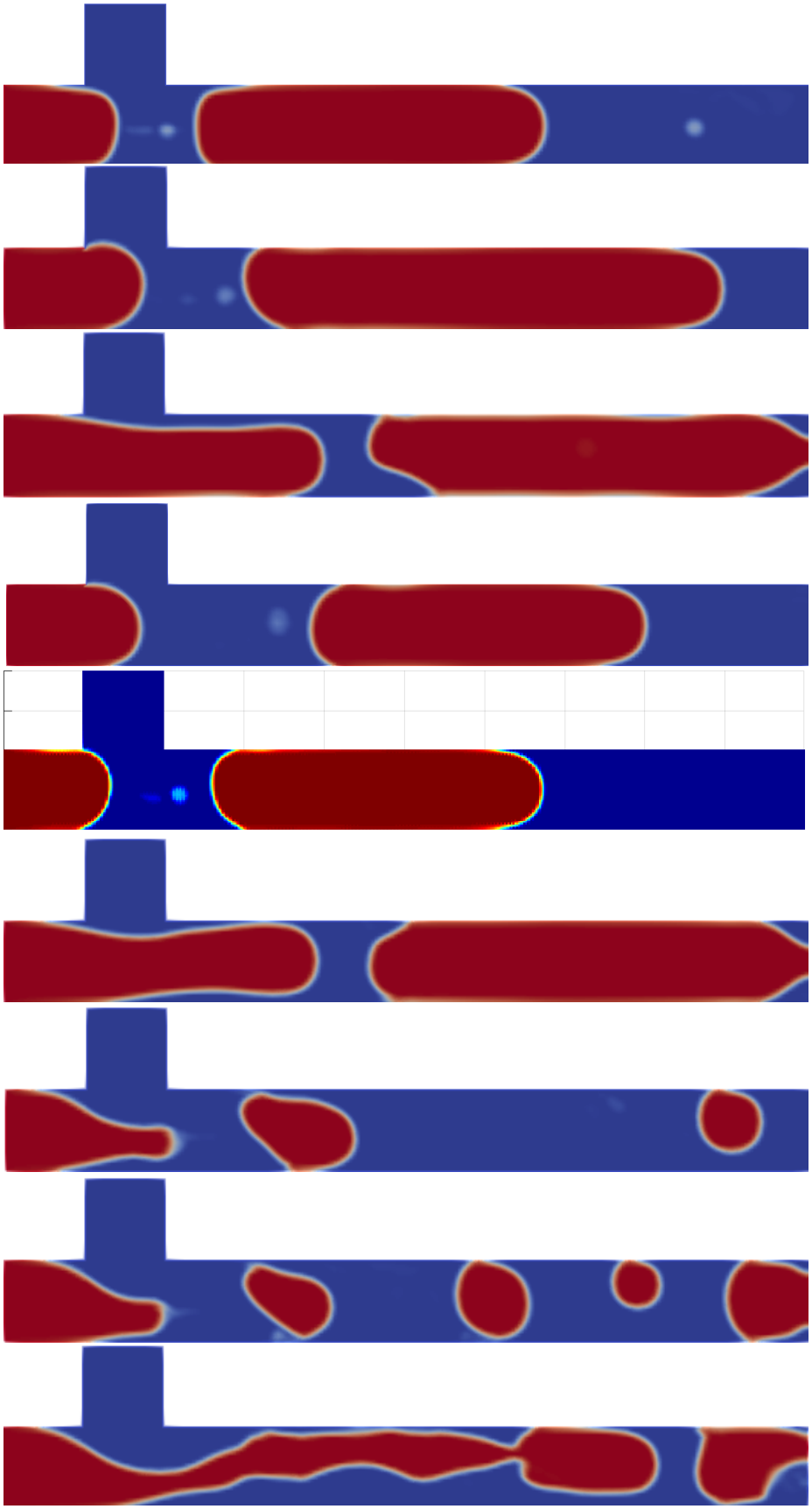


Figure 4.5: 7^{th} bubble release for combinations 1, 2, 3, 4, 6, 7, 8 and 9.

Figure 4.5 shows clear differentiations between the inlet velocity combinations. For starters, Combination 1 presents the slowest bubbles. When increasing a little the gas inlet velocity in Combination 2, the bubbles elongate because the liquid has trouble separating them at the same rate it did in Combination 1. Then, increasing five times the air inlet velocity in Combination 3, makes it impossible to show the whole bubble at its detachment. The separation between bubbles seems to be more abrupt than previous cases here. In Combination 4, the liquid inlet velocity is increased, which makes the bubbles shorter as it cuts them faster without deforming them. Combination 5 (see Figure 4.4) fastens the bubbles at the same time they elongate. Similar to Combination 3, when the gas inlet velocity is $0.5\,\mathrm{m/s}$ in the 6^{th} combination, the bubbles break more abruptly and do not maintain the round shape they have regularly in lower inlet velocities. The 7^{th} combination has the slowest gas and the fastest liquid velocities, which generate very small bubbles. From there, Combination 8 has still small bubbles (some smaller than others) but more frequent and differing in size when increasing the gas inlet velocity. And finally, when both inlet velocities are set to $0.5\,\mathrm{m/s}$ in the 9^{th} combination, the gas travels so fast that the liquid, even when being equally fast, takes longer to separate bubbles. It seems like when liquid tries to break a bubble, the gas has already gone too far.

4.2.1. Discussion

The final discussion can be made through the presented parameters of the 7^{th} bubble for each combination, as shown in Table 4.2 and Figures 4.6, 4.7, 4.8 and 4.9.

Velocity combination	U_{SG} (m/s)	U_{SL} (m/s)	f_B (Hz)	L_B (mm)	$U_{G,78}$ (m/s)	V_B (m^3)
1	0.05	0.05	44.4	7.10	0.50	$4.45 \cdot 10^{-9}$
2	0.1	0.05	51.3	6.60	0.50	$4.16 \cdot 10^{-9}$
3	0.5	0.05	81.6	9.20	0.80	$5.15 \cdot 10^{-9}$
4	0.05	0.1	52.6	6.30	0.57	$3.92 \cdot 10^{-9}$
5	0.1	0.1	59.7	2.60	0.40	$1.41 \cdot 10^{-9}$
6	0.5	0.1	114.3	7.00	1.00	$4.21 \cdot 10^{-9}$
7	0.05	0.5	148.1	1.40	0.80	$6.48 \cdot 10^{-10}$
8	0.1	0.5	333.3	0.80	0.80	$2.85 \cdot 10^{-10}$
9	0.5	0.5	307.7	1.70	1.33	$5.64 \cdot 10^{-10}$

Table 4.2: Parameter values for the different inlet velocities combinations

The parameters from Table 4.2 show values for one bubble, so we do not know the stability of these for each combination. Figure 4.6 illustrates how do parameter values change for the first seven bubbles.

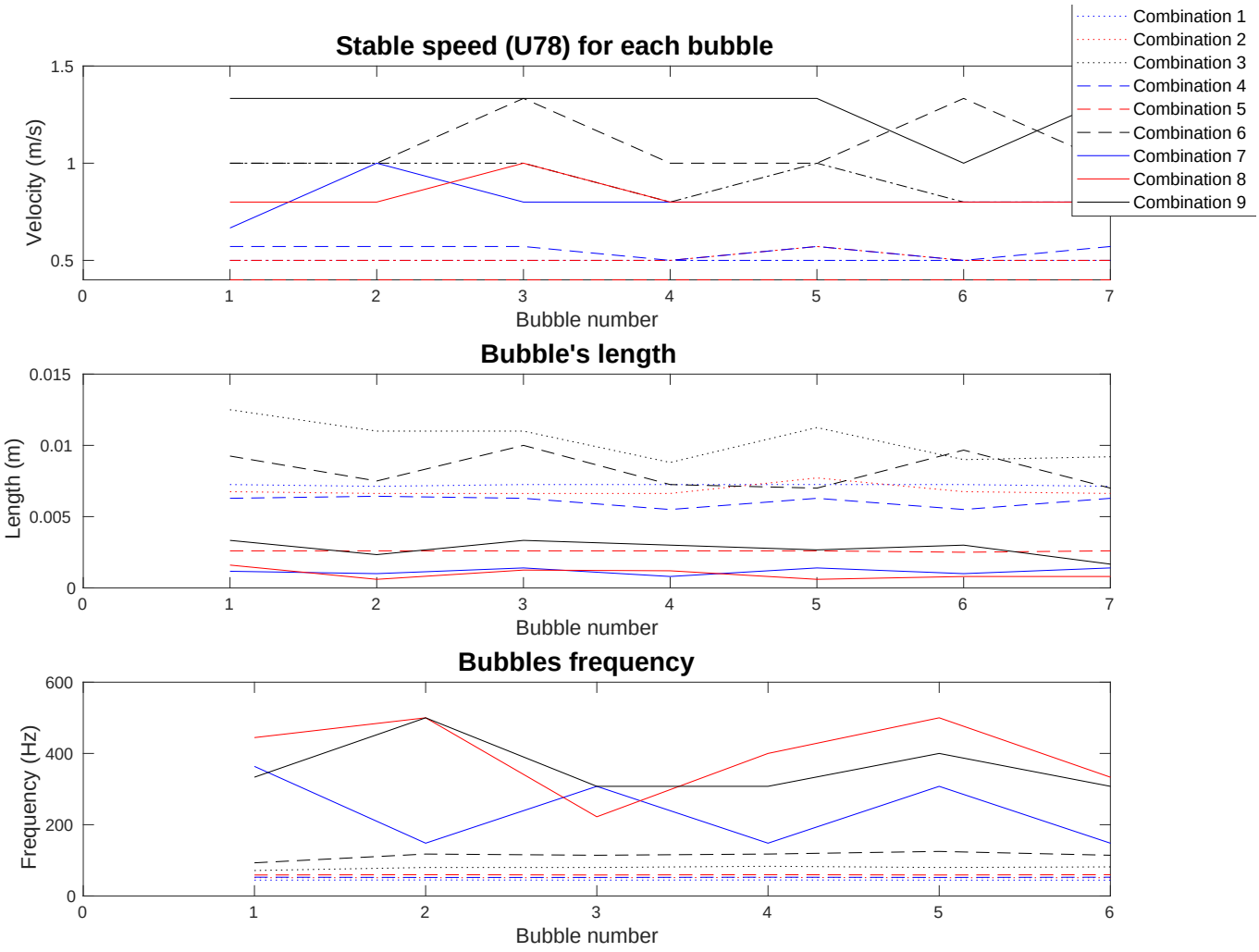


Figure 4.6: Parameters evolution per bubble for each combination of inlet velocities.

First of all, it can be seen in Table 4.2 that both the volume and the length of the bubble decrease when the liquid inlet velocity is highest (for combinations 7, 8 and 9). From 4.2, it seems like frequency increases noticeably when at least one of the inlet velocities is the highest tested value. Something similar happens with the bubble velocity, which increases greatly when one of inlet velocities is at $0.5\,\mathrm{m/s}$ and the other is higher or equal to $0.1\,\mathrm{m/s}$. When the liquid inlet velocity is higher, it becomes more difficult for the liquid to break the bubbles without deforming them a little.

Volume's parameter dependency on the bubble speed is what exempts its appearance in Figure 4.6. Both the bubble's velocity and length are quite stable for small inlet velocities. From Figure 4.6, the combinations with whether U_{SG} or U_{SL} equal to $0.5\,\mathrm{m/s}$ have higher variations between bubbles. The highest are presented in combinations 3 and 6 for length, and 3, 6, and 9 for velocity. The highest variations in the bubble frequency parameter are found in the 8^{th} and the 9^{th}, where U_{SL} is $0.5\,\mathrm{m/s}$.

To determine the behaviour of the parameters, we must predict their correlations through a set of equations or fittings. The parameter of frequency of the bubble presents the expected behaviour respect to the gas inlet velocity through the equation 4.1. The curves described in the equation cannot be drawn because we would need more points to plot

them correctly. However, points in Figure 4.7 present the correct trend of these[1].

$$f = f_{sat}(1 - e^{(-\frac{a_0}{f_{sat}} U_{SG})}) \tag{4.1}$$

As per the volume and the bubble velocity, using simple fittings from expressions 4.2 and 2.7, we can get the combination points behaviour in Figures 4.8 and 4.9.

$$\bar{V}_B = \frac{U_{SG}}{f \cdot L_{section}} \tag{4.2}$$

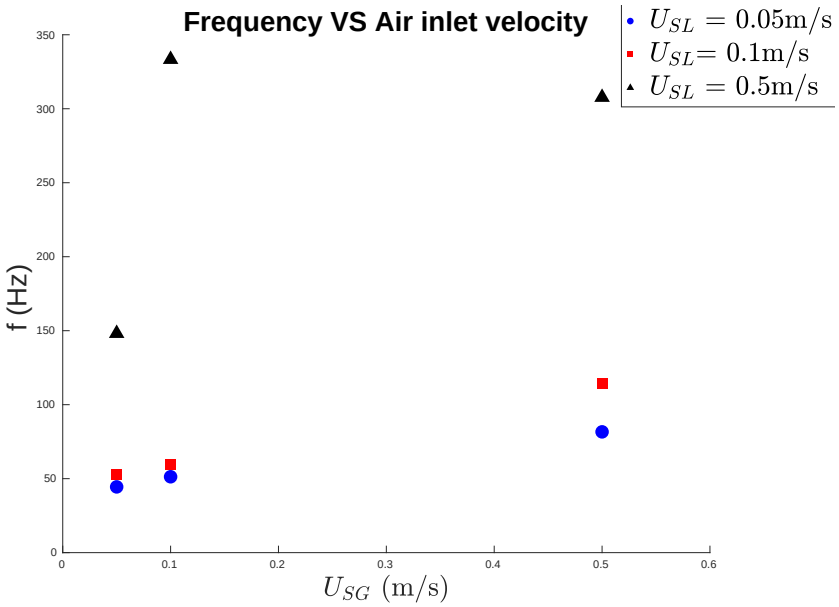


Figure 4.7: Frequency fitting dependence with the gas superficial inlet velocity.

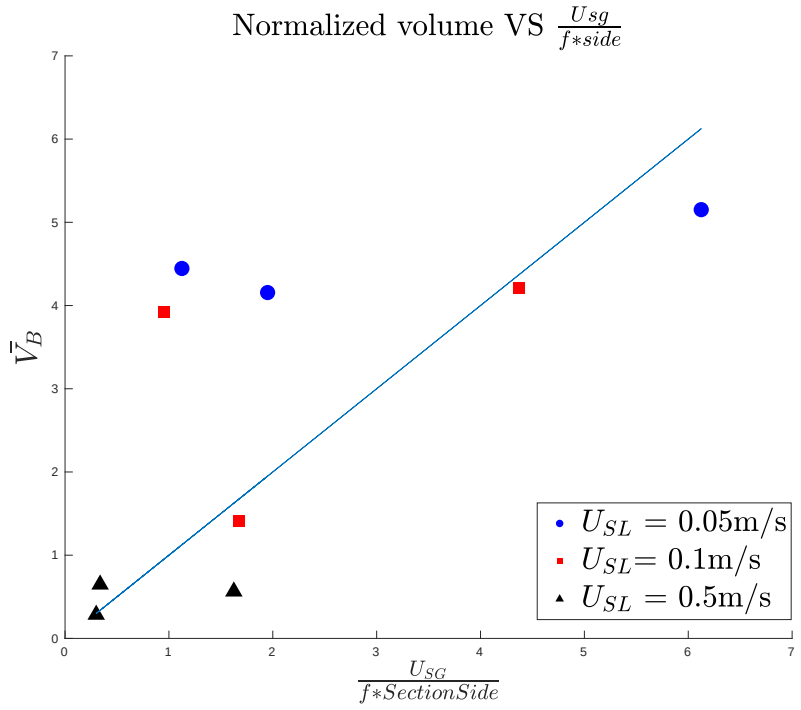


Figure 4.8: Normalized volume fitting dependence as a function of $\frac{U_{SG}}{f \cdot L}$[6].

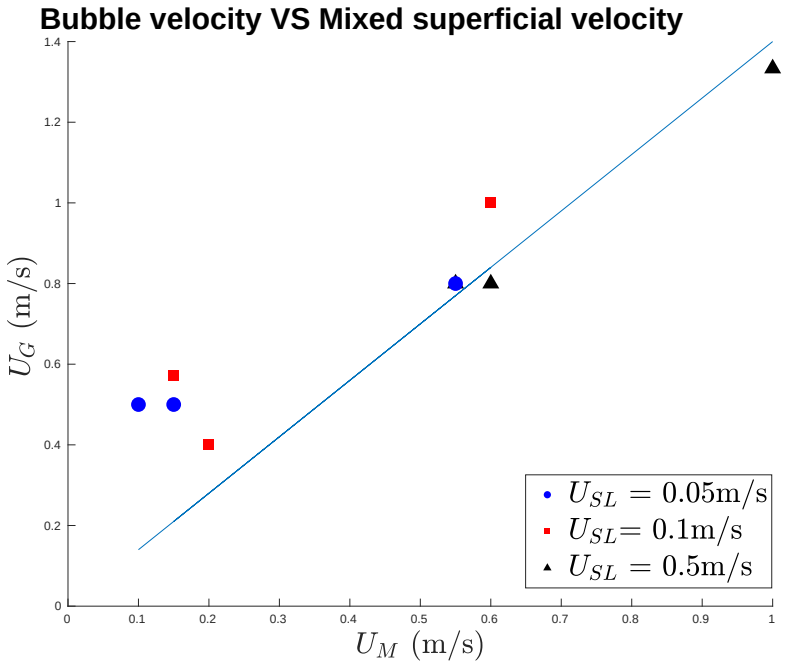


Figure 4.9: Bubble velocity fitting as a function of the mixed superficial velocity.

From Figure 4.7, the points for each liquid inlet velocity create a negative exponential curve shape, as expected[1]. However, the three curves cannot be drawn in our plot because three are very few points to define each of them correctly. If these were drawn either way, they would not be very close to their points due to the lack of accuracy.

Then Figure 4.8 presents a linear fitting for a normalized bubble volume as a function of $\frac{U_{SG}}{f_B \cdot L}$, which is an alternative way to compute the normalized volume. The normalized volume takes the bubble volume computed and divides it by 0.001 m from mesh dimensions. Generally, in 4.8, most of the points are very close to the linear fitting as we wanted, except for the three slowest combinations (1, 2 and 4).

Finally, Figure 4.9 presents a similar behaviour to the normalized volume, as all combinations are closer to the desired result so they are loyal to what was expected from these. Except for the slowest combinations too. The linear fitting line has the parameter C_0 as slope, which in this case is about 1.4. This is a little higher than what was obtained from the simulations and experimental values in [1], but we believe the difference is caused by the squared section mesh we used in this project. The overall value of C_0 for the 9 inlet velocity combinations is also a little lower than what we found in section 2.2.2. for Combination 5 ($U_{SL} = U_{SG} = 0.1\,\mathrm{m/s}$).

CONCLUSIONS

The goal of this project was to conduct numerical simulations in OpenFOAM and ParaView to generate bubbles in a T-junction mesh in microgravity conditions. The results obtained from these simulations were to be close to the ones in [1] and [15], or even improve their values by reducing the errors respect to their experimental results.

The methodology followed for the initial project's purpose was to generate the cylindrical mesh in a software external to OpenFOAM and transform it to a compatible version for the simulations to work properly. Then, the simulation conditions had to be established in OpenFOAM files to detach bubbles successfully. When these were made, samples of surfaces, points of the mesh and values of the evaluated fields had to be taken to process the collected data in Matlab. This was needed to discover which mesh, time step and contact angle were best to improve the performance of the simulations in the results section.

In this project, the methodology was strictly followed the way it is explained above. Nonetheless, we faced challenges along the way.

At first, the mesh was supposed to be progressive and refined at the junction where both air and water meet and generate the bubbles. Unfortunately, we experienced some drawbacks with the mesh generators and we spent a larger amount of time on them than what we were supposed to. In the end, we changed the geometry of the mesh to a squared-section T-junction to simplify this task and move on.

Then, when introducing the boundary conditions, the simulations produced a rare behaviour at the outlet of the mesh that endangered the reliability of the simulations results. Even though we invested time in solving this, no combination of conditions improved what we had initially, so we carried on with them. The strange behaviours obtained in the convergence tests of meshes, time steps and contact angles are due to this error. Also, that is why our results are further away from what we intended to obtain from the beginning. Still, what we learnt from varying the inlet velocities for both fluids is that when the liquid superficial velocity is increased, the bubbles detach sooner so they are more frequent and smaller. When the gas superficial velocity is increased, it generally produces the opposite effect. Other than that, the combination of both inlet velocities do not follow any strict rule that states which is the best for our simulations.

We also measured the evolution of pressure at four points around the junction in the results section. When evaluating this behaviour, we obtained similar results due to the closeness between the measuring points. Regardless of the closeness and the rare behaviour, the good news is that the pressure at the junction followed a logical physical behaviour.

For future projects, to improve the results we obtained, I think that the project's duration should be elongated. When organizing our schedule for a similar project, I believe it would be better to try less mesh generators to save some time that could be useful for solving the rare outlet conditions and obtain the desired physical behaviour. Then, the convergence tests would have to be more extensive in order to use the finest possible mesh and time step. The contact angle study should be extended as well in order to improve the results, and be closer to the experiment conclusions [1].

BIBLIOGRAPHY

[1] Montlaur A. Arias, S. Numerical study and experimental comparison of two-phase flow generation in a t-junction. Technical report, 2017. AIAA Journal, 12/2018, English. iii, v, 12, 14, 15, 16, 39, 41, 43, 52, 53, 54

[2] Gravity of earth - wikipedia. https://en.wikipedia.org/wiki/Gravity_of_Earth. On-line, 07/2019, English. 3

[3] Forbes business magazine. Why exploring space and investing in research is non-negotiable. https://www.forbes.com/sites/startswithabang/2017/10/26/even-while-the-world-suffers-investing-in-science-is-non-negotiable/#730b64e16470. On-line, 03/2019, English. 4

[4] NASA. What is microgravity? https://www.nasa.gov/audience/forstudents/5-8/features/nasa-knows/what-is-microgravity-58.html. On-line, 11/2018, English. 5

[5] NASA. Zero gravity facility tour. https://www.nasa.gov/specials/zero-g/. On-line, 11/2018, English. 6

[6] NASA Glenn Research Center. Zero gravity research facility. https://www1.grc.nasa.gov/facilities/zero-g/. On-line, 11/2018, English. 6

[7] California Science Center. Orion sounding rocket. On-line, 06/2019, English. 7

[8] NASA. What is a sounding rocket? https://www.nasa.gov/missions/research/f_sounding.html. On-line, 06/2019, English. 7

[9] Hajime Yano and JAXA. Microgravity geology: In-situ measurements and laboratory simulations. In *International Marco Polo Symposium and other Small Body Sample Return Missions*, 2009. PDF conference, 06/2019, English. 8

[10] UNOOSA. Russian research tests and missions. http://www.unoosa.org/pdf/pres/stsc2010/tech-13.pdf. On-line, 03/2019, English. 9

[11] (extrañas llamas a bordo de la estación espacial internacional - youtube. https://www.youtube.com/watch?time%5C_continue=176%5C&v=8eCTKIG9G-M. On-line, 11/2018, Spanish. 10

[12] HEMAV. Gravimav. https://hemav.com/gravimav/. On-line, 05/2019, Spanish. 10

[13] mitra UPC García Terrades, Arnau. Overview of the project "crear un entorno de microgravedad con un drone". https://mitra.upc.es/SIA/PFC_PUBLICA.DADES_PFC?w_codipfc=8723&v_curs_quad=2016-2. On-line, 05/2019, Spanish - Catalan - English. 10

[14] "The engineer" magazine article. "labyrinth" device sorts cancer cells from healthy blood the engineer. https://www.theengineer.co.uk/microfluidic-labyrinth-sorts-cancer-healthy-blood/. On-line, 05/2019, English. 11

[15] Blanca Dalfó Ferrer. Treball fi de grau - cfd study of the bubble generation process in a t-junction with inversed flows. https://upcommons.upc.edu/bitstream/handle/2117/121361/memoria.pdf?sequence=1&isAllowed=y. On-line, 01/2019, English. 14, 21, 54, 67, 115, 128

[16] Reynolds number - wikipedia. https://en.wikipedia.org/wiki/Reynolds_number. On-line, 07/2019, English. 16

[17] Onshape product design platform. https://www.onshape.com/. On-line, 03/2019, English. 17, 18

[18] Amina Bakkali Abderrahaman. Treball fi de grau - 3d cfd analysis of generation of bubbles in a double t-junction mini-channel. Technical report, 2018. Report, 05/2019, English. 21, 67, 115, 128

[19] KRUSS. Contact angle. https://www.kruss-scientific.com/services/education-theory/glossary/contact-angle/. On-line, 06/2019, English. 22

[20] OpenFOAMWiki. Interfoam. https://openfoamwiki.net/index.php/InterFoam. On-line, 06/2019, English. 23

[21] Openfoam user guide, version 6. http://foam.sourceforge.net/docs/Guides-a4/OpenFOAMUserGuide-A4.pdf. PDF User guide, 12/2018, English. 62

[22] Openfoam site user guide — contents. https://www.openfoam.com/documentation/user-guide/index.php. On-line, 12/2018, English. 62, 115

[23] Openfoam user guide: Cfd direct, architects of openfoam. https://cfd.direct/openfoam/user-guide/. On-line, 12/2018, English. 62, 115

[24] Solving files openfoam. https://www.openfoam.com/documentation/user-guide/solving.php. On-line, 03/2019, English. 76, 78

[25] Openfoam v6 user guide: 2.3 breaking of a dam. https://cfd.direct/openfoam/user-guide/v6-dambreak/. On-line, 05/2019, English. 78

[26] Openfoam v6 user guide: 3.4 running applications parallel. https://cfd.direct/openfoam/user-guide/v6-running-applications-parallel/. On-line, 04/2019, English. 79

[27] UCD PhilipCardiff. Introduction to meshing in openfoam. http://adhesion.ucd.ie/5th_OpenFOAM_User_Meeting/Home_files/PhilipCardiff-UCD-Introduction_to_Meshing_in_OpenFOAM.pdf. On-line, 01/2019, English. 86, 104, 105

[28] Scalar transport in t-junction pipe — simscale documentation. https://www.simscale.com/docs/content/validation/ScalarTransport/ScalarTransport.html. On-line, 03/2019, English. 86

[29] Standard boundary conditions. https://www.openfoam.com/documentation/user-guide/standard-boundaryconditions.php. On-line, 04/2019, English. 114, 115, 118

[30] Openfoam: User guide: Total pressure. https://www.openfoam.com/documentation/guides/latest/doc/guide-bcs-inlet-outlet-total-pressure.html. On-line, 05/2019, English. 115

APPENDICES

APPENDIX A. SIMULATION CASE

This chapter shows the commands to run simulations and captions of the case structure and files for further understanding. The commands and files shown are only correct for OpenFOAM v6 and ParaView v5.4, so these may change, depending on their software version. That is why commands from previous projects or forums have not worked well in this project.

A.1. Case structure

First, let's explain how the physical data from chapter 2 is organized in OpenFOAM cases and how do these work.

Each case contains all the information about one simulation. In this case, there should be at least the MSH file extension of the Fluent mesh from ANSYS, and three other folders (see Figure A.1). Additional files may be added because the computations have been made not in the virtual box platform but in the *UPC cluster ssh server*.

The file MSH can be read as a text file, called *"meshRect[number].msh"*. The other meshes do not appear here but can be transferred from Windows to Ubuntu through a shared directory.

The folders have always the same name so that OpenFOAM and ParaView can read their contents. Their names are *"0"*, *"constant"* and *"system"*. The most relevant data for the simulation is saved in these folders.

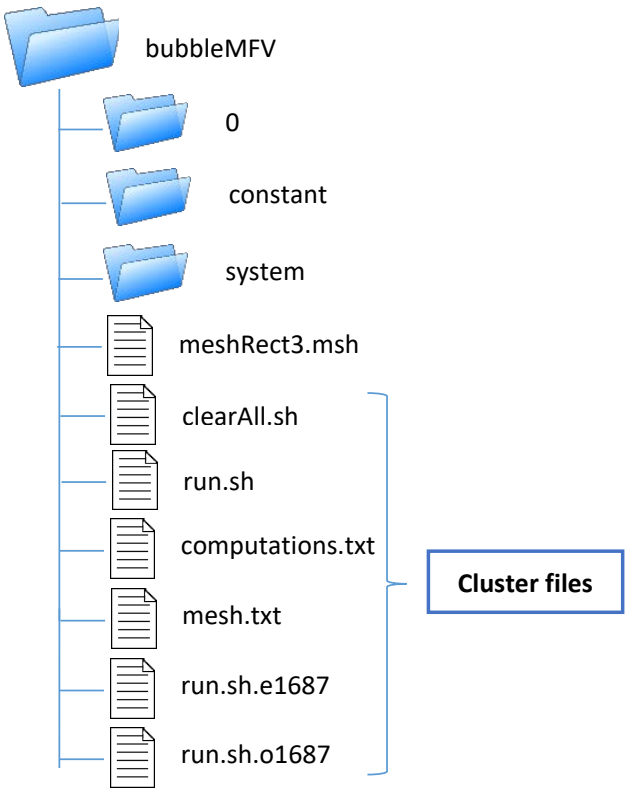


Figure A.1: Case *"bubbleMFV"* structure of folders and files.

In case "0" there are three Ubuntu text files, one per each main parameter: velocity, pressure and air fraction. These files, called *"U"*,"p_rgh" and *"alpha.gas"* respectively, contain the initial and boundary conditions as specified in sections 2.3.3. and 2.3.4.. The structure of the "0" case can be seen in Figure A.2

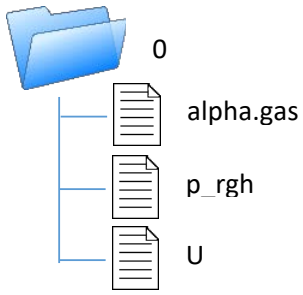


Figure A.2: "0" folder structure.

In "constant" case, the properties of the work regime are specified through three Ubuntu text files: *"g"*, *"transportProperties"* and *"turbulenceProperties"*. The "polyMesh" folder appears here as well when the mesh is created, saving its main features. The structure of the folder is seen in A.3.

File *"g"* defines gravity along the problem, which is 0 in all axis.

The *"transportProperties"* file defines which are the two phases in the simulation (gas and liquid), their Newtonian tranport model and their values of kinematic viscosity and density. At the end of the file, the value for the surface tension (σ) is established.

And the *"turbulenceProperties"* file declares the simulation as a laminar flow process.

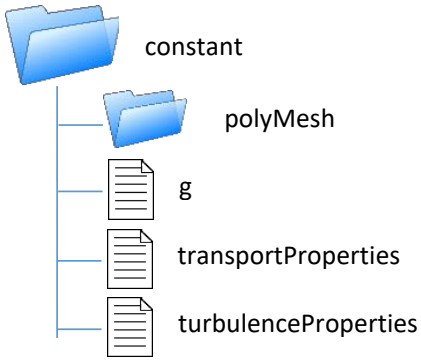


Figure A.3: "Constant" folder structure.

The "system" folder contains the files to start and control the simulation. It has at least the files *"controlDict"*, *"setFieldsDict"*, *"fvSchemes"* and *"fvSolution"*. Additional files may appear. The most common ones are the *"blockMeshDict"* and the *"decomposeParDict"*. Figure A.4 represents the structure of this folder.

The *"controlDict"* file contains the sampling functions, the initial and final time and other parameters like the writing interval to save simulation data. This one also states the solver used (*interFoam* in this case).

"SetFieldsDict" creates an initial region of liquid and gas in the mesh (ideally in each inlet, respectively) to initialize the problem.

"fvSchemes" describes the mathematical properties that the solver must use, from first and second derivatives method to gradient scheme or interpolation type.

Lastly, *"fvSolution"* specifies the solver how many cycles of correction there have to be, what kind of smoother and tolerance are needed, the relaxation factors in each field and the algorithm used in it. For *interFoam* solver, the common algorithm is PIMPLE.

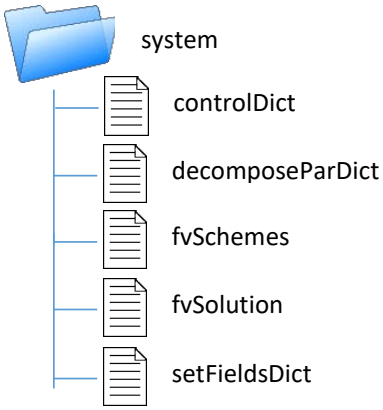


Figure A.4: "System" folder structure.

Once the simulation starts its calculations, new folders appear named after numbers. These have the names of its related time step and obtain information about the state of the main parameters (velocity, pressure and air fraction) at that time. These extra folders are needed to visualize the results in ParaView.

A.2. Case files

This section shows the files corresponding to the three folders of the simulation case: ”0”, ”constant” and ”system”. The commands and key words of these files might change in every OpenFOAM version. User guides regarding the version of the program installed have been used ([21], [22] and [23]).

A.2.1. ”0” folder

This folder imposes the initial and boundary conditions through three files: *alpha.gas* for the air fraction definition, *p_rgh* for the pressure scalar field and *U* for the velocity vectorial field in *(x,y,z)*.

To set the contact angle in *alpha.gas* file, the conditions of the face group ”walls2” (see section 2.3.2.1.) should be changed like in Figure A.6.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1806                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      alpha.gas;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];


internalField   uniform 0;

boundaryField
{
    gasinlet
    {
        type            fixedValue;
        value           uniform 1;
    }
    liquidinlet
    {
        type            fixedValue;
        value           uniform 0;
    }
    outlet
    {
        type            zeroGradient;
    }
    walls2
    {
        type            fixedValue;
        value           uniform 0;
    }
    walls1
    {
        type            fixedValue;
        value           uniform 0;
    }
}


// ************************************************************************* //
```

Figure A.5: Caption of the air fraction (alpha.gas) file.

```
        walls2
        {
            type            constantAlphaContactAngle;
            theta0          165; //theta0 = 180 - contactAngle
            limit           gradient;
            value           uniform 0;
        }
```

Figure A.6: Caption of the contact angle condition in the air fraction (alpha.gas) file.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p_rgh;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 -1 -2 0 0 0 0];

internalField   uniform 0; //initial state of the fluid, 0 means patm (pgage)

boundaryField
{
    gasinlet
    {
        type            zeroGradient;
    }

    liquidinlet
    {
        type            zeroGradient;
    }
    outlet
    {
        type            fixedValue;
        value           uniform 0; //Pgage
    }

    walls1
    {
        type            zeroGradient;
    }

    walls2
    {
        type            zeroGradient;
    }

}

// ************************************************************************* //
```

Figure A.7: Caption of the pressure (p_rgh) file.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0); //initial state of the fluid

boundaryField
{
    gasinlet
    {
        type            fixedValue;
        value           uniform (0.1 0 0); //(x y z) according to the mesh orientation
    }

    liquidinlet
    {
        type            fixedValue;
        value           uniform (0 0 -0.1); //(x y z) according to the mesh orientation
    }

    outlet
    {
        type            zeroGradient;
    }

    walls1
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    walls2
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

}

// ************************************************************************* //
```

Figure A.8: Caption of the velocity (U) file

A.2.2. ”Constant” folder

The ”constant” folder describes the fluid parameters and the work regime of the simulation through three other files: *g* for gravity conditions (neglected in this case), the *transportProperties* to define the main properties per fluid, and the *turbulenceProperties* to define the laminar flow.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       uniformDimensionedVectorField;
    location    "constant";
    object      g;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -2 0 0 0 0];
value   (0 0 0); //maximum value to consider microgravity is 10e-6. Rounded to 0

// ************************************************************************* //
```

Figure A.9: Caption of the gravity (g) file.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

phases (gas liquid);
gas
{
        transportModel  Newtonian;      //fluid work regime
        nu 8.163e-6;                    //kinematic viscosity of the air
        rho 1.225;                      //air density at SL
}
liquid
{
        transportModel  Newtonian;
        nu 1e-6;
        rho 1000;                       //water density
}

sigma 0.072; //surface tension

// ************************************************************************* //
```

Figure A.10: Caption of the transport properties file for each fluid.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

simulationType laminar;

// ************************************************************************* //
```

Figure A.11: Caption of the turbulence properties file for both fluids.

A.2.3. ”System” folder

The system folder contains 5 files: *controlDict*, *setFieldsDict*, *fvSchemes*, *fvSolution* and *decomposeParDict*.

The *controlDict* file (Figure A.12) is the longest file of the folder. It has the main control parameters of the simulation and the functions for sampling data.
The file presents the solver used (*application*) and the initial and final times in the simulations (*startTime* and *endTime*). The initial time in this case is set to *latestTime* to read which is the last time related folder in the case and starts the simulation from there. This is useful to continue a simulation when this one has been stopped abruptly and the data storage has been interrupted. If the cluster server went down for any reason, the simulation could be continued by writing just one command (”interFoam”).

The writing format of the stored data (*writeFormat* set in ASCII by default), the time step (*deltaT*) to maintain a desired Courant number and the *writeInterval* to write a time step folder are set in this first part of the file, too. The writing interval writes a folder every 1000 ticks of the computer clock time, or the cluster for that matter. These store parameters information at its correspondent time in a compressed format to save disk or cluster’s memory. Features like *timePrecision* and *runTimeModifiable* are set to the same values as [15] and [18].

The maximum Courant number is set to 2 instead of 1 because this needs to be higher than *maxAlphaCo* to make the simulation work better, according to www.cfd-online.com users advice. Otherwise, the time step had to be adjustable and a maximum time step had to be fixed to maintain the balance and not raise much the Courant number. In the end, it was preferred to keep a constant time step and use the trick of a higher *maxCo* limit.

The functions to sample data are divided following this order: probes, point and alpha data for the 7 sampled surfaces and then, the mean alpha value file per each sampled surface in the x-axis.

The probes use the *libsampling.so* function onto the given points of the mesh to save the values of the scalar field of pressure (*p_rgh*) every time step (*writeControl*). The same library is used for surface sampling.

Sampled surface of alpha values (3,4,6,7,8,9 mm and Y mid surface) are defined as *pointAndNormal planeType* surfaces to ensure including all their points and their *normalVector* direction. Inside the same sample folder, all surfaces have sampled values every 50 time steps (*writeControl*) to obtain 800 post processing files with the initial and final times set.

Then, the average air fraction value is sampled from the surfaces with a normal vector in the x-axis (3, 4, 6, 7, 8 and 9 millimetres). The function used was *libFieldFunctionObjects.so* and the values were sampled every time step. To sample values from surfaces, these need to be defined as in the previous case with a *pointAndNormal planeType*. *SampledSurfaceDict* was used to do this.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     interFoam;

startFrom       latestTime; //starts simulation from the last written time

startTime       0;

stopAt          endTime;

endTime         0.2;

deltaT          .000005;

writeControl    clockTime;

writeInterval   1000; //defines when to write a folder with the state of all parameters

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression yes;

timeFormat      general;

timePrecision   6;

runTimeModifiable yes;

adjustTimeStep  no;

maxCo           2;
maxAlphaCo      1;
//maxDeltaT     0.00005; //only applicable if adjustable time is on

functions
{
    //samples pressure of the 4 central points when forming a bubble
    probes
    {
        // Where to load it from
```

```
    functionObjectLibs ( "libsampling.so" );
    type            probes;

    // Name of the directory for probe data
    name            probes;

    // Write at same frequency as fields
    writeControl    timeStep;
    //writeInterval 1;

    // Fields to be probed
    fields
    (
        p_rgh
    );

    //Interpolation
    interpolationScheme     cellPoint;

    probeLocations
    (
        (0.000501092 -0.0005 0.000499999)   // point on the left (Y+)
        (0.0015 -0.0005 0.00123913)         // point at the top (Y+)
        (0.00250682 -0.0005 0.0005)         // point on the right (Y+)
        (0.00148303 -0.0005 0.00026087)     // point at the bottom (Y+)
    );
}


surfacesample
{
    type    surfaces;
    functionObjectLibs      ("libsampling.so");
    outputControl   timeStep;
    outputInterval  50;
    writeCompression yes;
    verbose         false;

    interpolationScheme     cellPoint;
    surfaceFormat           raw;

    surfaces
    (
            surfaceY
            {
                    type    cuttingPlane;
                    planeType   pointAndNormal;
                    pointAndNormalDict
                    {
                        basePoint (0.005 -0.0005 0.0005);
                        normalVector    (0 1 0);
                    }
                    interpolate true;
                    triangulate     false;
            }
```

```
surface3mm
{
        type    cuttingPlane;
        planeType   pointAndNormal;
        pointAndNormalDict
        {
            basePoint (0.003 -0.0005 0.0005);
            normalVector    (1 0 0);
        }
        interpolate true;
        triangulate     false;
}

surface4mm
{
        type    cuttingPlane;
        planeType   pointAndNormal;
        pointAndNormalDict
        {
            basePoint (0.004 -0.0005 0.0005);
            normalVector    (1 0 0);
        }
        interpolate true;
        triangulate     false;
}

surface6mm
{
        type    cuttingPlane;
        planeType   pointAndNormal;
        pointAndNormalDict
        {
            basePoint (0.006 -0.0005 0.0005);
            normalVector    (1 0 0);
        }
        interpolate true;
        triangulate     false;
}

surface7mm
{
        type    cuttingPlane;
        planeType   pointAndNormal;
        pointAndNormalDict
        {
            basePoint (0.007 -0.0005 0.0005);
            normalVector    (1 0 0);
        }
        interpolate true;
        triangulate     false;
}
```

```
            surface8mm
            {
                    type    cuttingPlane;
                    planeType   pointAndNormal;
                    pointAndNormalDict
                    {
                        basePoint (0.008 -0.0005 0.0005);
                        normalVector    (1 0 0);
                    }
                    interpolate true;
                    triangulate     false;
            }

            surface9mm
            {
                    type    cuttingPlane;
                    planeType   pointAndNormal;
                    pointAndNormalDict
                    {
                        basePoint (0.009 -0.0005 0.0005);
                        normalVector    (1 0 0);
                    }
                    interpolate true;
                    triangulate     false;
            }
    );
            fields (alpha.gas);
}

surface3mean //Air fraction average value
{
    type            surfaceFieldValue;
    libs            ("libfieldFunctionObjects.so");
    log             true;
    writeControl    timeStep;
    writeFields     false;
    regionType      sampledSurface;
    name            surface3mean;
    operation       average;
    fields (alpha.gas);
    surfaceFormat   dx;
    sampledSurfaceDict
    {
            type    cuttingPlane;
            planeType   pointAndNormal;
            pointAndNormalDict
            {
                basePoint (0.003 -0.0005 0.0005);
                normalVector    (1 0 0);
            }
            source          cells;
            interpolate     false;
            triangulate     false;
    }
}
```

```
surface4mean //Air fraction average value
{
    type            surfaceFieldValue;
    libs            ("libfieldFunctionObjects.so");
    log             true;
    writeControl    timeStep;
    writeFields     false;
    regionType      sampledSurface;
    name            surface4mean;
    operation       average;
    fields (alpha.gas);
    surfaceFormat   dx;
    sampledSurfaceDict
    {
            type    cuttingPlane;
            planeType   pointAndNormal;
            pointAndNormalDict
            {
                basePoint (0.004 -0.0005 0.0005);
                normalVector    (1 0 0);
            }
            source          cells;
            interpolate     false;
            triangulate     false;
    }
}

surface6mean //Air fraction average value
{
    type            surfaceFieldValue;
    libs            ("libfieldFunctionObjects.so");
    log             true;
    writeControl    timeStep;
    writeFields     false;
    regionType      sampledSurface;
    name            surface6mean;
    operation       average;
    fields (alpha.gas);
    surfaceFormat   dx;
    sampledSurfaceDict
    {
            type    cuttingPlane;
            planeType   pointAndNormal;
            pointAndNormalDict
            {
                basePoint (0.006 -0.0005 0.0005);
                normalVector    (1 0 0);
            }
            source          cells;
            interpolate     false;
            triangulate     false;
    }
}
```

```
surface7mean //Air fraction average value
{
    type            surfaceFieldValue;
    libs            ("libfieldFunctionObjects.so");
    log             true;
    writeControl    timeStep;
    writeFields     false;
    regionType      sampledSurface;
    name            surface7mean;
    operation       average;
    fields (alpha.gas);
    surfaceFormat   dx;
    sampledSurfaceDict
    {
            type    cuttingPlane;
            planeType   pointAndNormal;
            pointAndNormalDict
            {
                basePoint (0.007 -0.0005 0.0005);
                normalVector    (1 0 0);
            }
            source          cells;
            interpolate     false;
            triangulate     false;
    }
}

surface8mean //Air fraction average value
{
    type            surfaceFieldValue;
    libs            ("libfieldFunctionObjects.so");
    log             true;
    writeControl    timeStep;
    writeFields     false;
    regionType      sampledSurface;
    name            surface8mean;
    operation       average;
    fields (alpha.gas);
    surfaceFormat   dx;
    sampledSurfaceDict
    {
            type    cuttingPlane;
            planeType   pointAndNormal;
            pointAndNormalDict
            {
                basePoint (0.008 -0.0005 0.0005);
                normalVector    (1 0 0);
            }
            source          cells;
            interpolate     false;
            triangulate     false;
    }
}
```

```
    surface9mean //Air fraction average value
    {
        type            surfaceFieldValue;
        libs            ("libfieldFunctionObjects.so");
        log             true;
        writeControl    timeStep;
        writeFields     false;
        regionType      sampledSurface;
        name            surface9mean;
        operation       average;
        fields (alpha.gas);
        surfaceFormat   dx;
        sampledSurfaceDict
        {
                type    cuttingPlane;
                planeType   pointAndNormal;
                pointAndNormalDict
                {
                    basePoint (0.009 -0.0005 0.0005);
                    normalVector    (1 0 0);
                }
                source          cells;
                interpolate     false;
                triangulate     false;
        }
    }
}

// ************************************************************************* //
```

Figure A.12: Caption of the control dictionary (controlDict) file of the simulation.

The *setFieldsDict* file (Figure A.13) helps initialize the simulation by defining a volume of air in the *gasinlet* entrance ($\alpha = 1$). In *defaultFieldValues*, the air fraction value outside that volume is defined for water ($\alpha = 0$).

In *regions*, the volume of air at the initial time of simulation is defined. Its geometry is set to *boxToCell* because of the squared section of the mesh. The box volume of air is specified by two opposite vertices that cover the entire inlet section. The thickness of this box is determined by the distance between both vertices, which is about 0.5mm.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

defaultFieldValues
(
        volScalarFieldValue     alpha.gas 0
);

regions
(
        boxToCell
        {
                box (0 0 0) (0.0005 -0.001 0.001);
                fieldValues
                (
                        volScalarFieldValue     alpha.gas 1
                );
        }
);


// ************************************************************************* //
```

Figure A.13: Caption of the set fields dictionary (setFieldsDict) file of the simulation.

The *fvSchemes* dictionary file (Figure A.14) specify the numerical schemes in the solver equations during the computations. First and second time derivatives are defined in *ddtSchemes* field using Euler numerical schemes, while the gradient is defined as a non-steady process in *gradSchemes* through a numerical Gauss linear process. *DivSchemes* states the divergence numerical methods for parameters related to velocity and air fraction fields, mainly. Laplacian numerical schemes use a Gauss linear corrected approach to improve the values obtained from *laplacianSchemes*. The interpolation used in the simulation (field *interpolationSchemes*) is linear due to the simplicity of the problem, and the component of gradient normal to a cell face shall have a corrected numerical scheme in *snGradSchemes* section.

At the end, *fluxRequired* states which fields need a flux generation in the problem. In this case, the difference of pressure generates a flux so both *p_rgh* and *pcorr* are written in this field. The air fraction generates the major change in the flux, so it belong in it too. The velocity field is not written in it because it is only linked to the movement of the fluids along the mesh.

Further explanation of this file is available in [24].

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes //1st and 2nd derivatives
{
    default     Euler;
}
//non-steady process
gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    div(rhoPhi,U)      Gauss linearUpwind grad(U);
    div(phi,alpha)      Gauss vanLeer;
    div(phirb,alpha)    Gauss linear;
    div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

fluxRequired
{
        default no;
        p_rgh;
        pcorr;
        alpha.gas;
}
// ************************************************************************* //
```

Figure A.14: Caption of the *fvSchemes* dictionary file of the simulation.

Then, the *fvSolution* dictionary file (Figure A.15) controls the equations of the solver *interFoam*, their tolerances and the algorithms that this one uses. The PIMPLE algorithm is usually the one linked to the *interFoam* solver, so it is used in this project as well. The *relaxationFactors* section controls the technique used for stability improvement and convergence in computations, for both velocity and corrected pressure. This is called under-relaxation technique, which limits the changes in a variable from one iteration to the next.

Further information about this file composition is available in [24] and examples from similar cases are available in [25].

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    "alpha.gas.*"
    {
        nAlphaCorr      2;
        nAlphaSubCycles 1; //alpha equation solved in 1 half length time step
        cAlpha          1; //0 = no compression, 1 = conservative compression, N>1 = enhanced compression

        MULESCorr       yes;
        nLimiterIter    5;

        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-8;
        relTol          0;
    }

    "pcorr.*"
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-5;
        relTol          0;
    }

    p_rgh
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-07;
        relTol          0.05;
    }

    p_rghFinal
    {
        $p_rgh;
        relTol          0;
    }
```

```
    U
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-06;
        relTol          0;
    }
}

PIMPLE
{
    momentumPredictor   off;
    nOuterCorrectors    1;
    nCorrectors         3;
    nNonOrthogonalCorrectors 0;
    cAlpha                      1;
    nAlphaCorr                  2;
    nAlphaSubCycles             5;
}

relaxationFactors
{
        "U.*"            1; // 0.9 is more stable but 0.95 more convergent
        "pcorr.*"        1; // 0.9 is more stable but 0.95 more convergent
}


// ************************************************************************* //
```

Figure A.15: Caption of the *fvSolution* dictionary file of the simulation.

Finally, the *decomposeParDict* dictionary file (Figure A.16) divides the computations of the mesh in regions to run them in parallel and finish the calculations quicker. The *numberOf-Subdomains* defines the number of cores in which the mesh will be divided. In this project, the mesh should have been divided at least in two parts to treat the inlets separately.

The *method* used is the ”simple” one, so its coefficients were defined in *simpleCoeffs*. ”N” defines the parts of the mesh in which this is divided in all axes. Each part is computed and saved in a file called ”Processor X”, X being the number of the part, starting from 0. This division adds extra commands before and after the solver calculations, because the mesh must be divided and then unite results again.

The *hierarchicalCoeffs* define which parts are more urgent for calculations. In this project, the order is standard (x, y, z) and the priority is equal to all parts. All parts are computed in one disk to avoid the division in more than one PC physic processor.

This file is not used in this project because the cluster from the UPC was used and this one was fas enough already. Some tests took place initially to determine if it was necessary or not to use, though. Further information of this file is available in [26].

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      decomposeParDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

//I want to divide the mesh in two, as I need to treat the process of each inlet separately.
numberOfSubdomains      4;
method                  simple;

simpleCoeffs
{
        n       (2 1 2); //Inlets along x and z axis, so these have to be divided. Y is not included s
        delta   0.001; //Default value
}

hierarchicalCoeffs
{
        n       (1 1 1); //Same priority
        delta   0.001;
        order   xyz;    //most logical order
}

manualCoeffs
{
        dataFile        "";
}

distributed     no; //Just used in one disk.
roots           ();


// ************************************************************************* //
```

Figure A.16: Caption of the *decomposePar* dictionary file of the simulation.

A.3. Cluster files

Since the UPC cluster is used for computations in this project, it generates four additional files in our case when sending the simulations there. To run a simulation, the ”run.sh” file is used (explained in section A.4.).

The first file generated when running a simulation is the ”mesh.txt”. This file saves the mesh properties when this one is created. Instead of showing this in a terminal screen, the information is saved here. See Figure A.17.

When computing, the terminal screen data of the calculation process is saved in the ”computations.txt” (See Figure A.18).

And then, two extra files are generated which are not specified by the user: ”run.sh.eNjob” and ”run.sh.oNjob”. The first file writes the errors produced when a simulation (or a job)

stops, while the second one writes the operations of the process. These are shown in Figures A.19 and A.20.

```
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1806                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
Build  : v1806
Arch   : "LSB;label=32;scalar=64"
Exec   : fluent3DMeshToFoam meshRect3.msh
Date   : May 23 2019
Time   : 14:29:53
Host   : "clufa02"
PID    : 41012
I/O    : uncollated
Case   : /cluster/users/students/grosado/bubbleAC15
nProcs : 1
trapFpe: Floating point exception trapping enabled (FOAM_SIGFPE).
fileModificationChecking : Monitoring run-time modified files using timeStampMaster (fileModificationSkew 10)
allowSystemOperations : Allowing user-supplied system call operations

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

Dimension of grid: 3
Number of points: 145728
Number of faces: 412045
Number of cells: 133308
PointGroup: 3 start: 0 end: 121483.  Reading points...done.
PointGroup: 4 start: 121484 end: 145727.  Reading points...done.
FaceGroup: 1 start: 0 end: 387802.  Reading uniform faces...done.
FaceGroup: 5 start: 387803 end: 388331.  Reading uniform faces...done.
FaceGroup: 6 start: 388332 end: 388860.  Reading uniform faces...done.
FaceGroup: 7 start: 388861 end: 389389.  Reading uniform faces...done.
FaceGroup: 8 start: 389390 end: 410986.  Reading uniform faces...done.
FaceGroup: 9 start: 410987 end: 412044.  Reading uniform faces...done.
CellGroup: 2 start: 0 end: 133307 type: 1
Zone: 1 name: interior-meshrect-freeparts_part_1 type: interior.  Reading zone data...done.
Zone: 2 name: meshrect-freeparts_part_1 type: fluid.  Reading zone data...done.
Zone: 5 name: gasinlet type: velocity-inlet.  Reading zone data...done.
Zone: 6 name: liquidinlet type: velocity-inlet.  Reading zone data...done.
Zone: 7 name: outlet type: pressure-outlet.  Reading zone data...done.
Zone: 8 name: walls2 type: wall.  Reading zone data...done.
Zone: 9 name: walls1 type: wall.  Reading zone data...done.

FINISHED LEXING

Creating patch 0 for zone: 5 name: gasinlet type: velocity-inlet
Creating patch 1 for zone: 6 name: liquidinlet type: velocity-inlet
Creating patch 2 for zone: 7 name: outlet type: pressure-outlet
Creating patch 3 for zone: 8 name: walls2 type: wall
Creating patch 4 for zone: 9 name: walls1 type: wall
Creating cellZone 0 name: meshrect-freeparts_part_1 type: fluid
Creating faceZone 0 name: interior-meshrect-freeparts_part_1 type: interior
faceZone from Fluent indices: 0 to: 387802 type: interior
patch 0 from Fluent indices: 387803 to: 388331 type: velocity-inlet
patch 1 from Fluent indices: 388332 to: 388860 type: velocity-inlet
patch 2 from Fluent indices: 388861 to: 389389 type: pressure-outlet
patch 3 from Fluent indices: 389390 to: 410986 type: wall
patch 4 from Fluent indices: 410987 to: 412044 type: wall

Writing mesh to "/cluster/users/students/grosado/bubbleAC15/constant/region0"

End
```

Figure A.17: File ”mesh.txt” from cluster.

```
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1806                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
Build  : v1806
Arch   : "LSB;label=32;scalar=64"
Exec   : interFoam
Date   : May 23 2019
Time   : 14:29:58
Host   : "clufa02"
PID    : 41014
I/O    : uncollated
Case   : /cluster/users/students/grosado/bubbleAC15
nProcs : 1
trapFpe: Floating point exception trapping enabled (FOAM_SIGFPE).
fileModificationChecking : Monitoring run-time modified files using timeStampMaster (fileModificationSkew 10)
allowSystemOperations : Allowing user-supplied system call operations

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

Create mesh for time = 0


PIMPLE: Operating solver in PISO mode

Reading field p_rgh

Reading field U

Reading/calculating face flux field phi

Reading transportProperties

Selecting incompressible transport model Newtonian
Selecting incompressible transport model Newtonian
Selecting turbulence model type laminar
Selecting laminar stress model Stokes

Reading g

Reading hRef
Calculating field g.h

No MRF models present

No finite volume options present
DICPCG:  Solving for pcorr, Initial residual = 1, Final residual = 8.8384e-06, No Iterations 193
time step continuity errors : sum local = 8.03491e-10, global = -2.95496e-12, cumulative = -2.95496e-12
Courant Number mean: 0.0208197 max: 0.0703984

Starting time loop

--> FOAM Warning :
    From function void Foam::timeControl::read(const Foam::dictionary&)
    in file db/functionObjects/timeControl/timeControl.C at line 103
    Reading "/cluster/users/students/grosado/bubbleAC15/system/controlDict.functions.surfacesample"
    Using deprecated 'outputControl'
    Please use 'writeControl' with 'writeInterval'
Reading surface description:
    surfaceY
    surface3mm
    surface4mm
    surface6mm
    surface7mm
    surface8mm
    surface9mm

Courant Number mean: 0.0208197 max: 0.0703984
Interface Courant Number mean: 0 max: 0
Time = 5e-06

PIMPLE: iteration 1
```

```
smoothSolver:  Solving for alpha.gas, Initial residual = 1, Final residual = 8.82367e-13, No Iterations 2
Phase-1 volume fraction = 4.54545e-05  Min(alpha.gas) = 0  Max(alpha.gas) = 1
MULES: Correcting alpha.gas
MULES: Correcting alpha.gas
Phase-1 volume fraction = 4.54545e-05  Min(alpha.gas) = -1.18123e-11  Max(alpha.gas) = 1
DICPCG:  Solving for p_rgh, Initial residual = 1, Final residual = 0.0492522, No Iterations 157
time step continuity errors : sum local = 5.12778e-06, global = 9.93343e-10, cumulative = 9.90388e-10
DICPCG:  Solving for p_rgh, Initial residual = 0.00041988, Final residual = 1.92004e-05, No Iterations 152
time step continuity errors : sum local = 5.87315e-07, global = -7.82513e-09, cumulative = -6.83474e-09
DICPCG:  Solving for p_rgh, Initial residual = 2.62262e-05, Final residual = 9.8892e-08, No Iterations 149
time step continuity errors : sum local = 2.91486e-09, global = -1.4221e-10, cumulative = -6.97695e-09
ExecutionTime = 3.99 s  ClockTime = 5 s

Courant Number mean: 0.0145776 max: 0.0703455
Interface Courant Number mean: 8.72878e-05 max: 0.0703455
Time = 1e-05

PIMPLE: iteration 1
smoothSolver:  Solving for alpha.gas, Initial residual = 0.258087, Final residual = 2.75716e-10, No Iterations 2
Phase-1 volume fraction = 6.26769e-05  Min(alpha.gas) = -0.0653561  Max(alpha.gas) = 1
MULES: Correcting alpha.gas
MULES: Correcting alpha.gas
Phase-1 volume fraction = 6.26833e-05  Min(alpha.gas) = -0.059539  Max(alpha.gas) = 1
DICPCG:  Solving for p_rgh, Initial residual = 0.00349816, Final residual = 0.000171379, No Iterations 157
time step continuity errors : sum local = 5.0709e-06, global = -1.02922e-09, cumulative = -8.00617e-09
DICPCG:  Solving for p_rgh, Initial residual = 0.0280702, Final residual = 0.00136993, No Iterations 38
time step continuity errors : sum local = 4.86689e-07, global = -2.61794e-08, cumulative = -3.41855e-08
DICPCG:  Solving for p_rgh, Initial residual = 0.00146535, Final residual = 8.82993e-08, No Iterations 176
time step continuity errors : sum local = 3.05845e-11, global = -9.95784e-14, cumulative = -3.41856e-08
ExecutionTime = 5.56 s  ClockTime = 7 s

Courant Number mean: 0.0144233 max: 0.0757027
Interface Courant Number mean: 6.28262e-05 max: 0.0386139
Time = 1.5e-05

PIMPLE: iteration 1
smoothSolver:  Solving for alpha.gas, Initial residual = 0.159054, Final residual = 1.75439e-10, No Iterations 2
Phase-1 volume fraction = 7.97594e-05  Min(alpha.gas) = -0.13201  Max(alpha.gas) = 1
```

Figure A.18: File ”computations.txt” from cluster.

```
/var/spool/gridengine/execd/clufa02/job_scripts/1658: línea 8: unlimit: orden no encontrada

```

Figure A.19: File ”run.sh.eNjob” generated by the cluster.

```
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1806                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
Build  : v1806
Arch   : "LSB;label=32;scalar=64"
Exec   : setFields
Date   : May 23 2019
Time   : 14:29:56
Host   : "clufa02"
PID    : 41013
I/O    : uncollated
Case   : /cluster/users/students/grosado/bubbleAC15
nProcs : 1
trapFpe: Floating point exception trapping enabled (FOAM_SIGFPE).
fileModificationChecking : Monitoring run-time modified files using timeStampMaster (fileModificationSkew 10)
allowSystemOperations : Allowing user-supplied system call operations

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

Create mesh for time = 0

Reading setFieldsDict

Setting field default values
    Setting internal values of volScalarField alpha.gas

Setting field region values
    Adding faces with centre within boxes 1((0 0 0) (0.0005 -0.001 0.001))
    Setting patchField values of volScalarField alpha.gas

End
```

Figure A.20: File ”run.sh.oNjob” generated by the cluster.

A.4. Commands

In this project, a UPC cluster has been used to perform the computations quicker. The commands used are the same whether it is on a cluster or on a local PC. The difference remains in how these are claimed.

A.4.1. Running a simulation in a local PC

The laptop used for this project runs simulations in an Ubuntu platform from the virtual box. The commands must be written from the terminal in the case directory. To change from home directory in the terminal, the following command must be used.

```
cd SimulationCase_Name
```

When the files of the case are complete (section A.2.), we write the following commands in order to run the simulation.

```
fluent3DMeshToFoam meshRect3.msh   //Converts ANSYS mesh to an OpenFOAM
    compatible format
setFields              //Initializes the simulation
interFoam              //Initializes the solver and starts the calculations
paraFoam               //Opens ParaView to visualize the computations of the
    solver
```

In case the *decomposeParDict* is used in a simulation, the commands change slightly as shown below.

```
fluent3DMeshToFoam meshRect3.msh
setFields
decomposePar             //Creates folders for each core/processor
interFoam              //Starts the calculations in each core file
recomposePar             //Unites all core folders information and generates
    time step stored data as in usual simulations
paraFoam
```

A.4.2. Running a simulation in the cluster

To run this same simulation in the cluster, a *bash* file must be created with the extension ".sh". These are quite common on Ubuntu, and they are created by introducing in a regular blank text file a set of commands to convert the file and some more from A.4.1. The cluster user provided by the university does include OpenFOAM software installed but not ParaView, so the steps are:

1.- Copy the simulation case from the local PC to the cluster with one command.

```
scp -r ./SimulationCase_Name grosado@147.83.7.212:Case_GivenName
```

The local directory was "home/user", so the access through terminal could be simplified with a point, while **grosado** is my username assigned from the university. The cluster and the local PC can have different names for the same case if wanted.

2.- Once inside the cluster case directory, only one command is needed to run the bash file. All bash files were called ”run.sh” in this case.

```
qsub run.sh    //Adds the specified job of the bash file to the cluster queue
```

In ”run.sh”, the commands to convert the text file to a bash document must be written at the heading of the file before writing anything else.

```
#!/bin/bash
#
#$ -cwd
#$ -S /bin/bash
#
#$ -q allnodes
```

To run simulations in the cluster, only the following commands are written in the job file (”run.sh”). Some instructions must save the processes in a file because the cluster does not print these in a terminal nor screen to indicate its correct working.

```
unlimit -s 50000 //Not necessary but it does ensure there is enough free space in the cluster for the computations

fluent3DMeshToFoam meshRect3.msh >mesh.txt
setFields
interFoam >computations.txt
```

3.- When the job is done, the case must be copied back to the local PC directory.

```
scp -r grosado@147.83.7.212:/cluster/users/students/grosado/Case_GivenName ./SimulationCase_Name
```

4.- Eventually, when the case is back to the local PC, we visualize the results by writing ”paraFoam” on the terminal to open ParaView. We can also open Matlab by writing ”matlab” on terminal to work the post processing of the simulation.

In the cluster, neither the *decomposePar* and *reconstructPar* commands work. Somehow, these commands are not recognized in the cluster, so there are no exceptions in this case.

The bright side of the cluster though, is that this one can compute several simulations at once without slowing down any process. That is not possible in an ordinary local PC.

APPENDIX B. MESH GENERATION

In this section of the appendix, there will examples of how the meshes or the files from OpenFOAM tools looked like. As it is explained in 2.3.2.1., the geometry of the mesh was changed at some point to simplify the problem, but the goal here is showing part of the work behind the mesh generation process before it came to that.

The programs and tools used to generate meshes were:

- *blockMesh* from OpenFOAM.
- GMSH program.
- *snappyHexMesh* from OpenFOAM.
- www.onshape.com only to export geometries.
- ANSYS.

B.1. BlockMesh tool

The *blockMesh* tool was used initially to generate the mesh in OpenFOAM, following instructions from documents like [27] for simpler meshes. At first, a cylinder-shaped mesh was generated to try the tool, and it worked just fine. However, generate a T-junction requires multiple interconnected vertices, which added extra difficulty to the process. Some samples of files for a cylindrical T-junction can be seen in Figures B.1, B.2 and B.3 to show the progress made in each file.

Figure B.1 shows the initial attempt of generating a T-junction mesh. This one has some errors related to the orientation of the faces written in the *boundary* section.

Figure B.2 shows the following version of the T-junction mesh formed in *blockMesh*. The errors of orientation from the first attempt were fixed and a new association of the geometry is made, through smaller blocks in cylindrical pipes. This new association was supposed to save some lines in the file, but that did not generate the mesh we wanted at all, so we had to go to write a third version of the file.

In the end, Figure B.3 was the final attempt to generate a mesh with this tool. The mesh in this case was reorganized as there were already too many vertices to connect while the tool had problems uniting both cylindrical pipes in a junction. To realise where was the problem, we divided the mesh in 4 parts when creating it: the *gasInlet* region, the *liquidInlet* and vertical pipe region, the joint region and the rest of the horizontal pipe until the *outlet*. Then, it came to realization that the problem was at the junction and that this tool was not made to generate two pipes sharing a joint like in this case. Even when the errors were fixed, ParaView always showed a twisted mesh far from what we intended, which was the result of *blockMesh* putting together the different parts of the mesh oddly.

According to www.cfd-online.com users, *blockMesh* must be used for simple meshes with one block at most, meaning no junctions could be made. Also, when searching similar examples like [28], it was seen that users tend to use other programs like ANSYS or

GMSH for a quick complex mesh generation, or generate complex geometries with *snappyHexMesh* tool from OpenFOAM.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.001; //then I can write dimensions in mm
vertices
(
        //FRONT CIRCLE (y,z)
        (0 0 0)         //0, center
        (0 0 0.5)       //1, top
        (0 -0.5 0)      //2, left
        (0 0 -0.5)      //3, bottom
        (0 0.5 0)       //4, right

        //CIRCLE FROM THE BACK (y,z)
        (10 0 0)        //5, center
        (10 0 0.5)      //6, top
        (10 0.5 0)      //7, right
        (10 0 -0.5)     //8, bottom
        (10 -0.5 0)     //9, left

        //TOP CIRCLE (x,y)
        (1.5 0 1.5)     //10, center
        (2 0 1.5)       //11, back
        (1.5 -0.5 1.5)  //12, left
        (1 0 1.5)       //13, front
        (1.5 0.5 1.5)   //14, right

        //INTERSECTION POINTS (x,y,z)
        (1.5 0 0)       //15, center
        (1 0 0.5)       //16, front up point
        (1.5 -0.5 0)    //17, left down point
        (2 0 0.5)       //18, back up point
        (1.5 0.5 0)     //19, right down point

        //CENTRAL CIRCLE (outlet orientation) -V
        //15 is center
        //17 is left
        //19 is right
        (1.5 0 0.5)     //20, top middle
        (1.5 0 -0.5)    //21, bottom middle
```

```
        //CENTRAL CIRCLE (liquid inlet orientation)
        //20 is center
        //16 is front
        //18 is back
        (1.5 -0.5 0.5)  //22, left
        (1.5 0.5 0.5)   //23, right

        //CENTRAL CIRCLE (gasInlet orientation) -F1
        (1 0 0)         //24, center
        //16 is top
        (1 -0.5 0)      //25, left
        (1 0 -0.5)      //26, bottom
        (1 0.5 0)       //27, right

        //CENRAL CIRCLE (outlet orientation) -F2
        (2 0 0)         //28, center
        //18 is top
        (2 0.5 0)       //29, right
        (2 0 -0.5)      //30, bottom
        (2 -0.5 0)      //31, left
);
blocks
(
        //front pack
        hex (1 2 0 0 25 24 24 16) (200 5 5) simpleGrading (3 1 1) //block 0
        hex (2 3 0 0 26 24 24 25) (200 5 5) simpleGrading (3 1 1)
        hex (3 4 0 0 27 24 24 26) (200 5 5) simpleGrading (3 1 1)
        hex (4 1 0 0 16 24 24 27) (200 5 5) simpleGrading (3 1 1)

        //back pack
        hex (6 5 5 9 31 18 28 28) (300 5 5) simpleGrading (0.3 1 1) //block 4
        hex (9 5 5 8 30 31 28 28) (300 5 5) simpleGrading (0.3 1 1)
        hex (8 5 5 7 29 30 28 28) (300 5 5) simpleGrading (0.3 1 1)
        hex (7 5 5 6 18 29 28 28) (300 5 5) simpleGrading (0.3 1 1)

        //verticalBlock
        hex (11 12 10 10 22 20 20 18) (200 5 5) simpleGrading (3 1 1) //block 8
        hex (12 13 10 10 16 20 20 22) (200 5 5) simpleGrading (3 1 1)
        hex (13 14 10 10 23 20 20 16) (200 5 5) simpleGrading (3 1 1)
        hex (14 11 10 10 18 20 20 23) (200 5 5) simpleGrading (3 1 1)

        //centralBlock
        hex (16 25 24 24 31 28 28 18) (300 10 10) simpleGrading (1 1 1) //block 12
        hex (25 26 24 24 30 28 28 31) (300 10 10) simpleGrading (1 1 1)
        hex (26 27 24 24 29 28 28 30) (300 10 10) simpleGrading (1 1 1)
        hex (27 16 24 24 18 28 28 29) (300 10 10) simpleGrading (1 1 1)
);
edges
(

        arc 1 2 (0 -0.707107 0.707107)
        arc 2 3 (0 -0.707107 -0.707107)
        arc 3 4 (0 0.707107 -0.707107)
        arc 4 1 (0 0.707107 0.707107)
```

```
        arc 6 7 (10 0.707107 0.707107)
        arc 7 8 (10 0.707107 -0.707107)
        arc 8 9 (10 -0.707107 -0.707107)
        arc 9 6 (10 -0.707107 0.707107)

        arc 11 12 (2.207107 -0.707107 1.5)
        arc 12 13 (0.792893 -0.707107 1.5)
        arc 13 14 (0.792893 0.707107 1.5)
        arc 14 11 (2.207107 0.707107 1.5)

        arc 16 17 (0.792893 -0.707107 0.25)
        arc 17 18 (2.207107 -0.707107 0.25)
        arc 18 19 (2.207107 0.707107 0.25)
        arc 19 16 (0.792893 0.707107 0.25)
);

boundary
(
        gasInlet
        {
                type patch;
                faces
                (
                        (1 2 0 0)
                        (2 3 0 0)
                        (3 4 0 0)
                        (4 1 0 0)
                );
        }

        liquidInlet
        {
                type patch;
                faces
                (
                        (11 12 10 10)
                        (12 13 10 10)
                        (13 14 10 10)
                        (14 11 10 10)
                );
        }

        outlet
        {
                type patch;
                faces
                (
                        (6 7 5 5)
                        (7 8 5 5)
                        (8 9 5 5)
                        (9 6 5 5)
                );
        }
```

```
walls1
{
        type patch;
        faces
        (
                (16 23 14 13)
                (13 12 22 16)
                (11 14 23 18)
                (12 11 18 22)
        );
}

walls2
{
        type patch;
        faces
        (
                (1 4 27 16)
                (4 3 26 27)
                (3 2 25 26)
                (2 1 16 25)

                (7 6 18 19)
                (8 7 29 30)
                (30 31 9 8)
                (31 18 6 9)

                (18 29 27 16)
                (30 29 27 26)
                (26 25 31 30)
                (25 16 18 31)
        );
}

others
{
        type patch;
        faces
        (
                (16 23 20 20) //walls1 (blocks 8-11)
                (22 16 20 20)
                (18 22 20 20)
                (23 18 20 20)

                (10 14 23 20)
                (10 13 16 20)
                (10 12 22 20)
                (10 11 18 20)


                (16 25 24 24) //walls2 (blocks 0-3)
                (25 26 24 24)
                (26 27 24 24)
                (27 16 24 24)
```

```
                        (0 1 16 24)
                        (0 2 25 24)
                        (0 3 26 24)


                        (31 18 28 28) //walls2 (blocks 4-7)
                        (18 29 28 28)
                        (29 30 28 28)
                        (30 31 28 28)

                        (28 18 6 5)
                        (28 31 9 5)
                        (28 30 8 5)
                        (28 29 7 5)

                        (24 16 18 28) //walls2 (blocks 12-15)
                        (24 25 31 28)
                        (24 26 30 28)
                        (24 27 29 28)
                );
        }
);

mergePatchPairs
(
        (walls1 walls2)
);

// ************************************************************************* //
```

Figure B.1: *BlockMeshDict* file, first T-junction attempt.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.001; //then I can write dimensions in mm
vertices
(
        //FRONT CIRCLE (y,z)
        (0 0 0)         //0, center
        (0 0 0.5)       //1, top
        (0 -0.5 0)      //2, left
        (0 0 -0.5)      //3, bottom
        (0 0.5 0)       //4, right

        //CIRCLE FROM THE BACK (y,z)
        (10 0 0)        //5, center
        (10 0 0.5)      //6, top
        (10 0.5 0)      //7, right
        (10 0 -0.5)     //8, bottom
        (10 -0.5 0)     //9, left

        //TOP CIRCLE (x,y)
        (1.5 0 1.5)     //10, center
        (2 0 1.5)       //11, back
        (1.5 -0.5 1.5)  //12, left
        (1 0 1.5)       //13, front
        (1.5 0.5 1.5)   //14, right

        //INTERSECTION POINTS (x,y,z)
        (1.5 0 0)       //15, center
        (1 0 0.5)       //16, front up point
        (1.5 -0.5 0)    //17, left down point
        (2 0 0.5)       //18, back up point
        (1.5 0.5 0)     //19, right down point

        //CENTRAL CIRCLE (outlet orientation)
        //15 is center
        //17 is left
        //19 is right
        (1.5 0 0.5)     //20, top middle
        (1.5 0 -0.5)    //21, bottom middle
);
blocks
(
        //horizontalBlock WHOLE (front part)
```

```
        hex (1 2 0 0 9 5 5 6) (200 10 10) simpleGrading (1 1 1) //block 0
        hex (2 3 0 0 8 5 5 9) (200 10 10) simpleGrading (1 1 1)
        hex (3 4 0 0 7 5 5 8) (200 10 10) simpleGrading (1 1 1)
        hex (4 1 0 0 6 5 5 7) (200 10 10) simpleGrading (1 1 1)


        //horizontalBlock (back part)
        //hex (20 17 15 15 9 5 5 6) (200 10 10) simpleGrading (1 1 1) //block 4
        //hex (17 21 15 15 8 5 5 9) (200 10 10) simpleGrading (1 1 1)
        //hex (21 19 15 15 7 5 5 8) (200 10 10) simpleGrading (1 1 1)
        //hex (19 20 15 15 6 5 5 7) (200 10 10) simpleGrading (1 1 1)

        //verticalBlock
        hex (11 12 10 10 17 15 15 18) (100 20 20) simpleGrading (1 1 1)
        hex (12 13 10 10 16 15 15 17) (100 20 20) simpleGrading (1 1 1)
        hex (13 14 10 10 19 15 15 16) (100 20 20) simpleGrading (1 1 1)
        hex (14 11 10 10 18 15 15 19) (100 20 20) simpleGrading (1 1 1)
);
edges
(
        arc 1 2 (0 -0.707107 0.707107)
        arc 2 3 (0 -0.707107 -0.707107)
        arc 3 4 (0 0.707107 -0.707107)
        arc 4 1 (0 0.707107 0.707107)

        arc 6 7 (10 0.707107 0.707107)
        arc 7 8 (10 0.707107 -0.707107)
        arc 8 9 (10 -0.707107 -0.707107)
        arc 9 6 (10 -0.707107 0.707107)

        arc 11 12 (2.207107 -0.707107 1.5)
        arc 12 13 (0.792893 -0.707107 1.5)
        arc 13 14 (0.792893 0.707107 1.5)
        arc 14 11 (2.207107 0.707107 1.5)

        arc 16 17 (0.792893 -0.707107 0.25)
        arc 17 18 (2.207107 -0.707107 0.25)
        arc 18 19 (2.207107 0.707107 0.25)
        arc 19 16 (0.792893 0.707107 0.25)
);

boundary
(
        gasInlet
        {
                type patch;
                faces
                (
                        (1 2 0 0)
                        (2 3 0 0)
                        (3 4 0 0)
                        (4 1 0 0)
                );
        }
```

```
liquidInlet
{
        type patch;
        faces
        (
                (11 12 10 10)
                (12 13 10 10)
                (13 14 10 10)
                (14 11 10 10)
        );
}

outlet
{
        type patch;
        faces
        (
                (6 7 5 5)
                (7 8 5 5)
                (8 9 5 5)
                (9 6 5 5)
        );
}

walls1
{
        type wall;
        faces
        (
                (19 18 15 15)
                //(11 14 19 18)
                //(11 12 18 17)
                (18 17 15 15)
                //(12 13 17 16)
                (17 16 15 15)
                //(13 14 16 19)
                (16 19 15 15)
        );

}

walls2
{
        type wall;
        faces
        (
                (1 4 19 16)
                (18 19 7 6)
                (4 3 19 21)
                (19 21 8 7)
                (2 1 16 17)
                (17 18 6 9)
                (3 2 17 21)
                (21 17 9 8)
        );
}
```

```
        others
        {
                type empty;
                faces
                (
                );
        }
);

mergePatchPairs
(
        //(walls1 mergewall)
        //(walls2 mergewall)
);

// ************************************************************************* //
```

Figure B.2: *BlockMeshDict* file, modifications to the first T-junction attempt.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.001; //then I can write dimensions in mm
vertices
(
        //FRONT CIRCLE (y,z)
        (0 0 0)         //0, center
        (0 0 0.5)       //1, top
        (0 -0.5 0)      //2, left
        (0 0 -0.5)      //3, bottom
        (0 0.5 0)       //4, right

        //CIRCLE FROM THE BACK (y,z)
        (10 0 0)        //5, center
        (10 0 0.5)      //6, top
        (10 0.5 0)      //7, right
        (10 0 -0.5)     //8, bottom
        (10 -0.5 0)     //9, left

        //TOP CIRCLE (x,y)
        (1.5 0 1.5)     //10, center
        (2 0 1.5)       //11, back
        (1.5 -0.5 1.5)  //12, left
        (1 0 1.5)       //13, front
        (1.5 0.5 1.5)   //14, right

        //INTERSECTION POINTS (x,y,z)
        (1.5 0 0)       //15, center
        (1 0 0.5)       //16, front up point
        (1.5 -0.5 0)    //17, left down point
        (2 0 0.5)       //18, back up point
        (1.5 0.5 0)     //19, right down point

        //CENTRAL CIRCLE (outlet orientation) -V
        //15 is center
        //17 is left
        //19 is right
        (1.5 0 0.5)     //20, top middle
        (1.5 0 -0.5)    //21, bottom middle

        //CENTRAL CIRCLE (liquid inlet orientation)
        //20 is center
        //16 is front
        //18 is back
        (1.5 -0.5 0.5)  //22, left
```

```
        (1.5 0.5 0.5)   //23, right

        //CENTRAL CIRCLE (gasInlet orientation) -F1
        (1 0 0)         //24, center
        //16 is top
        (1 -0.5 0)      //25, left
        (1 0 -0.5)      //26, bottom
        (1 0.5 0)       //27, right

        //CENRAL CIRCLE (outlet orientation) -F2
        (2 0 0)         //28, center
        //18 is top
        (2 0.5 0)       //29, right
        (2 0 -0.5)      //30, bottom
        (2 -0.5 0)      //31, left

);
blocks
(
        //front pack
        hex (1 2 0 0 25 24 24 16) (200 5 5) simpleGrading (3 1 1) //block 0
        hex (2 3 0 0 26 24 24 25) (200 5 5) simpleGrading (3 1 1)
        hex (3 4 0 0 27 24 24 26) (200 5 5) simpleGrading (3 1 1)
        hex (4 1 0 0 16 24 24 27) (200 5 5) simpleGrading (3 1 1)


        //back pack
        hex (6 5 5 9 31 18 28 28) (300 5 5) simpleGrading (0.3 1 1) //block 4
        hex (9 5 5 8 30 31 28 28) (300 5 5) simpleGrading (0.3 1 1)
        hex (8 5 5 7 29 30 28 28) (300 5 5) simpleGrading (0.3 1 1)
        hex (7 5 5 6 18 29 28 28) (300 5 5) simpleGrading (0.3 1 1)

        //verticalBlock
        hex (11 12 10 10 22 20 20 18) (200 5 5) simpleGrading (3 1 1) //block 8
        hex (12 13 10 10 16 20 20 22) (200 5 5) simpleGrading (3 1 1)
        hex (13 14 10 10 23 20 20 16) (200 5 5) simpleGrading (3 1 1)
        hex (14 11 10 10 18 20 20 23) (200 5 5) simpleGrading (3 1 1)

        //centralBlock
        hex (16 25 24 24 31 28 28 18) (300 10 10) simpleGrading (1 1 1) //block 12
        hex (25 26 24 24 30 28 28 31) (300 10 10) simpleGrading (1 1 1)
        hex (26 27 24 24 29 28 28 30) (300 10 10) simpleGrading (1 1 1)
        hex (27 16 24 24 18 28 28 29) (300 10 10) simpleGrading (1 1 1)
);
edges
(
        arc 1 2 (0 -0.707107 0.707107)
        arc 2 3 (0 -0.707107 -0.707107)
        arc 3 4 (0 0.707107 -0.707107)
        arc 4 1 (0 0.707107 0.707107)

        arc 6 7 (10 0.707107 0.707107)
        arc 7 8 (10 0.707107 -0.707107)
        arc 8 9 (10 -0.707107 -0.707107)
        arc 9 6 (10 -0.707107 0.707107)
```

```
        arc 11 12 (2.207107 -0.707107 1.5)
        arc 12 13 (0.792893 -0.707107 1.5)
        arc 13 14 (0.792893 0.707107 1.5)
        arc 14 11 (2.207107 0.707107 1.5)

        arc 16 17 (0.792893 -0.707107 0.25)
        arc 17 18 (2.207107 -0.707107 0.25)
        arc 18 19 (2.207107 0.707107 0.25)
        arc 19 16 (0.792893 0.707107 0.25)
);

boundary
(
        gasInlet
        {
                type patch;
                faces
                (
                        (1 2 0 0)
                        (2 3 0 0)
                        (3 4 0 0)
                        (4 1 0 0)
                );
        }

        liquidInlet
        {
                type patch;
                faces
                (
                        (11 12 10 10)
                        (12 13 10 10)
                        (13 14 10 10)
                        (14 11 10 10)
                );
        }

        outlet
        {
                type patch;
                faces
                (
                        (6 7 5 5)
                        (7 8 5 5)
                        (8 9 5 5)
                        (9 6 5 5)
                );
        }

        walls1
        {
                type patch;
                faces
                (
                        (16 23 14 13)
                        (13 12 22 16)
                        (11 14 23 18)
                        (12 11 18 22)
                );
```

```
    }

    walls2
    {
            type patch;
            faces
            (
                    (1 4 27 16)
                    (4 3 26 27)
                    (3 2 25 26)
                    (2 1 16 25)

                    (7 6 18 19)
                    (8 7 29 30)
                    (30 31 9 8)
                    (31 18 6 9)

                    (18 29 27 16)
                    (30 29 27 26)
                    (26 25 31 30)
                    (25 16 18 31)
            );
    }

    others
    {
            type patch;
            faces
            (
                    (16 23 20 20) //walls1 (blocks 8-11)
                    (22 16 20 20)
                    (18 22 20 20)
                    (23 18 20 20)

                    (10 14 23 20)
                    (10 13 16 20)
                    (10 12 22 20)
                    (10 11 18 20)


                    (16 25 24 24) //walls2 (blocks 0-3)
                    (25 26 24 24)
                    (26 27 24 24)
                    (27 16 24 24)

                    (0 1 16 24)
                    (0 2 25 24)
                    (0 3 26 24)


                    (31 18 28 28) //walls2 (blocks 4-7)
                    (18 29 28 28)
                    (29 30 28 28)
                    (30 31 28 28)
```

```
                        (28 18 6 5)
                        (28 31 9 5)
                        (28 30 8 5)
                        (28 29 7 5)

                        (24 16 18 28) //walls2 (blocks 12-15)
                        (24 25 31 28)
                        (24 26 30 28)
                        (24 27 29 28)
                );
        }
);

mergePatchPairs
(
        (walls1 walls2)
);

// ************************************************************************* //
```

Figure B.3: *BlockMeshDict* file, last T-junction attempt.

B.2. GMSH programme

The GMSH programme was one of the easiest programmes to use. To generate the mesh geometry, it was only necessary to indicate the dimensions of both cylinders and use the option ”unite” to create their junction. The mesh generation was more complex, though. The only things that could be defined from the mesh were its style (if it goes from one face to the other or if it is contained in a volume) and its density at some point of the geometry, which allowed progressive mesh generation at the joint.

The downside of the program was that the cells shape could not be chosen at all with our geometry, as hexagonal cells were only available for simpler geometries. Sometimes a few cells forced the conversion of their shape to hexagonal, but that created more problems for the simulation as the surrounding cells needed to adjust their geometry to cover the rest of the mesh. To do that, very small angles were produced in several cells while others were just too small to adapt their geometry, which increased insanely the Courant number. The solver never ended the computations, it was always interrupted first.

The first mesh generated in this program was made in millimetres instead of using the international system of units as I should have. Also the first attempt had a very long horizontal pipe while the rest got cut every attempt for simplicity and spare time. Images of the first meshes generated appear in pictures B.4, B.5 and B.6. These were refined at the joint region, so it is hard to differentiate the cells. The face groups were assigned in GMSH to visualize them in OpenFOAM as in Figure B.7.

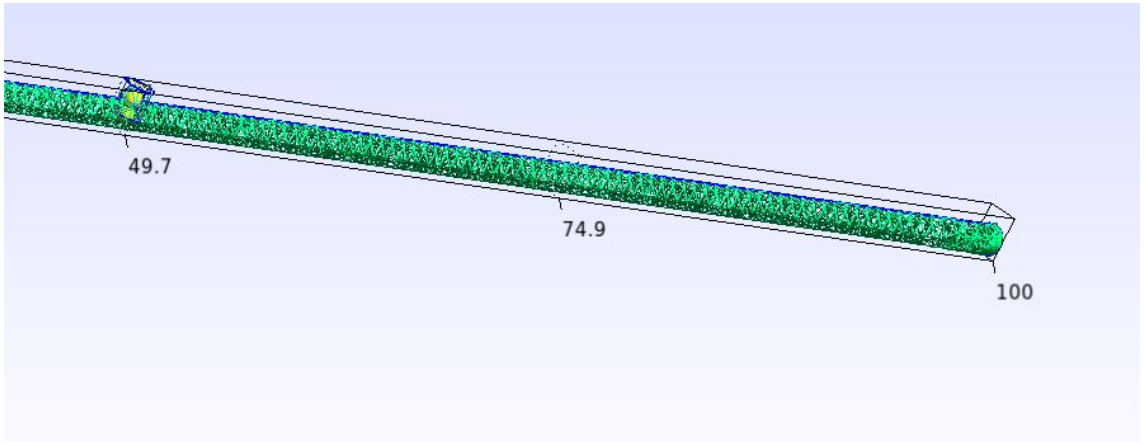


Figure B.4: First mesh created in GMSH.

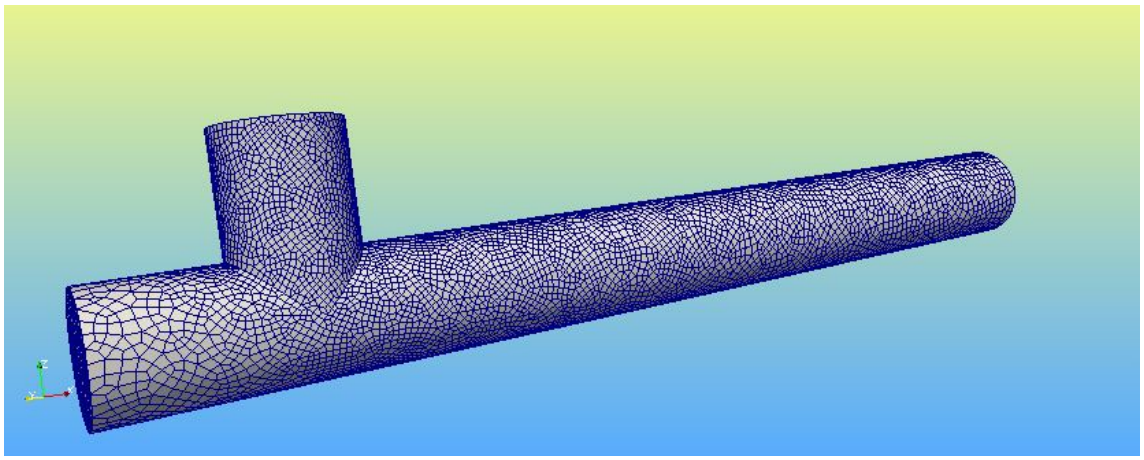


Figure B.5: Shorter second mesh created in GMSH with new *liquidInlet* localization.

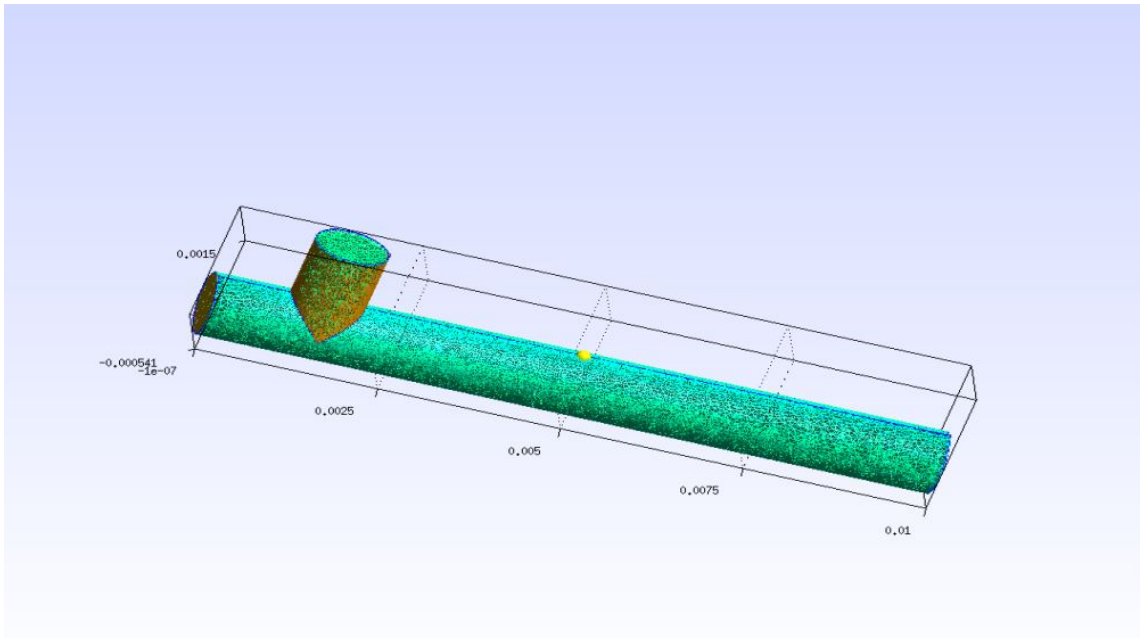


Figure B.6: Fourth mesh created in GMSH, this time with new dimensions.

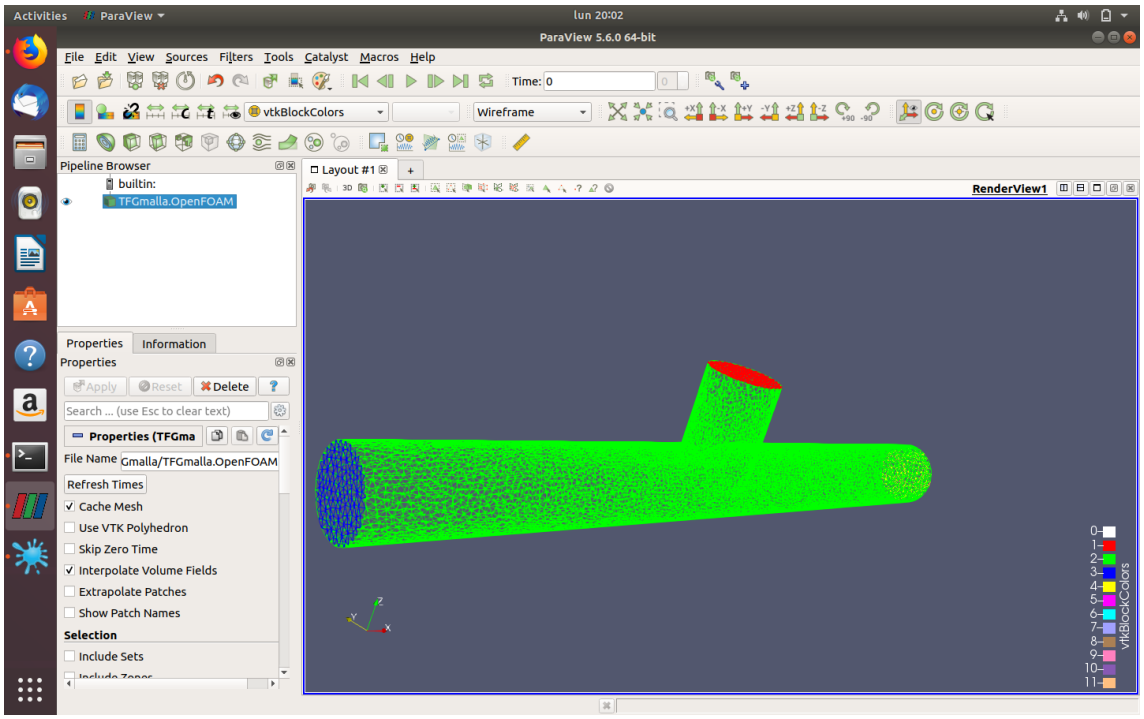


Figure B.7: Differentiated patches of the first mesh seen in ParaView, once converted.

The meshes produced in this program were quite a few, as shown in the stand-alone files of Figure B.8. GMSH generates the geometry and the mesh separately in the program, so even though these are always dependant from one another, GMSH outputs two files with each extension: ".geo" and ".msh" for geometry and mesh, respectively. The difference between both files contents are shown in Figures B.9 and B.10

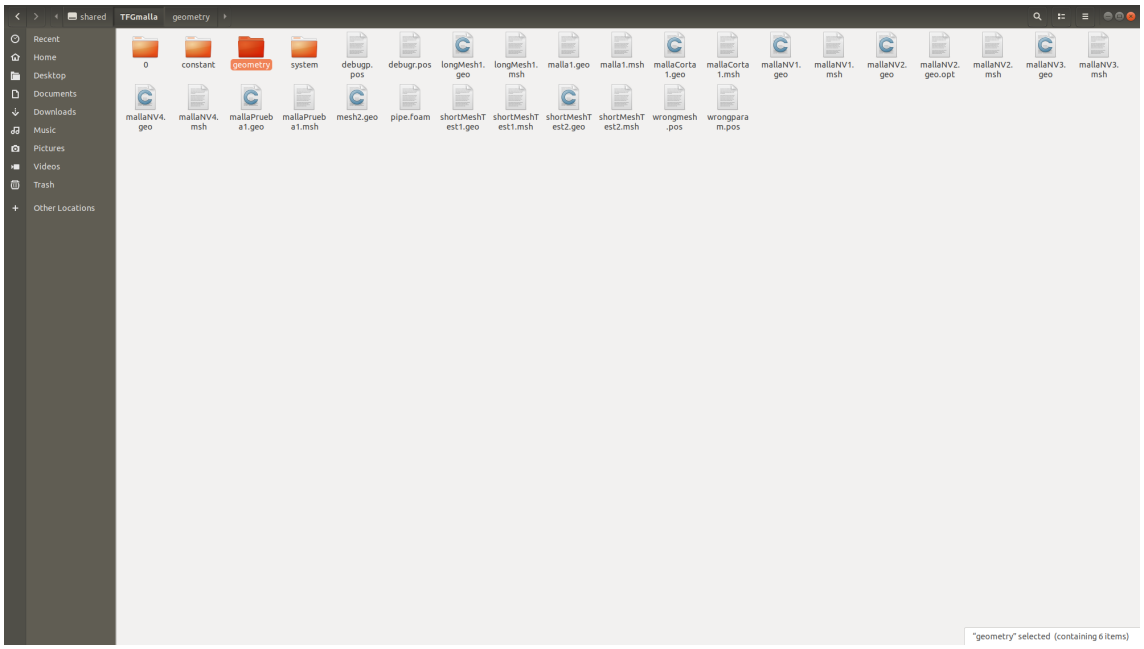


Figure B.8: Most of the GMSH files (".geo" and ".msh") generated in this project.

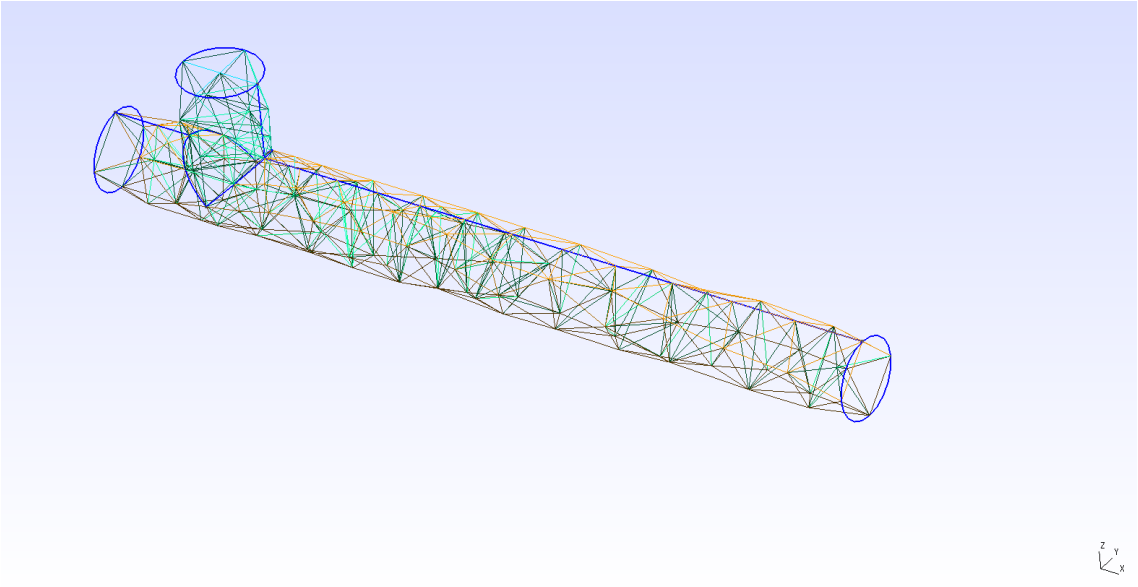


Figure B.9: ”.geo” extension of the GMSH generated mesh.

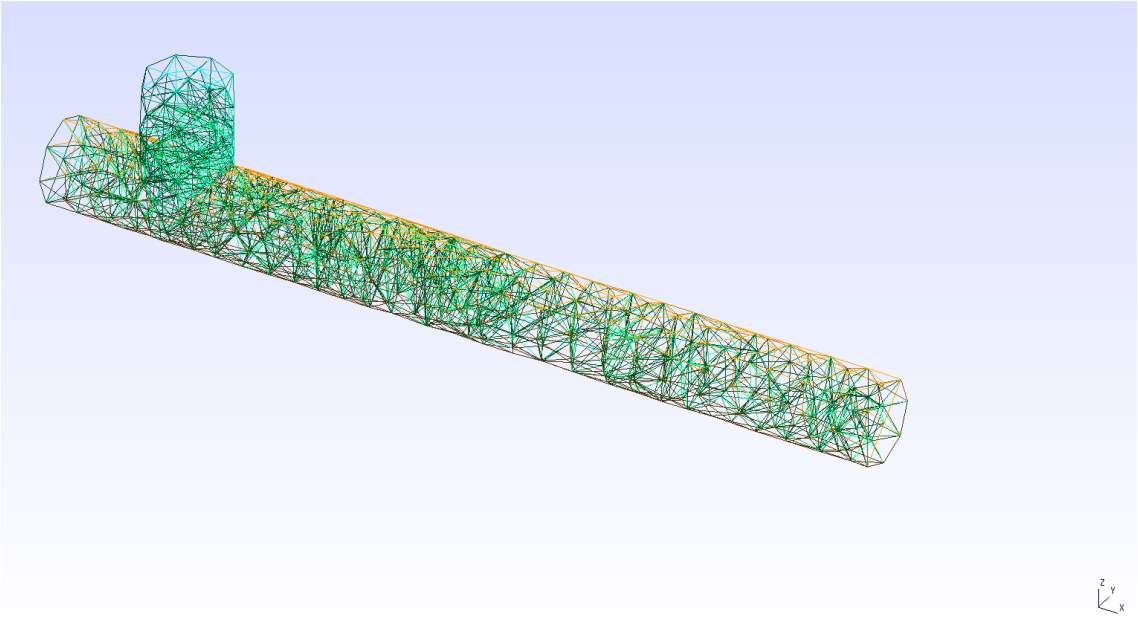


Figure B.10: ”.msh” extension of the GMSH generated mesh.

For OpenFOAM, it was enough to convert the file with the extension ”.msh” to work, using the *gmshToFoam nameMesh.msh* command. Figure B.10 shows the real data the mesh as specified by the user, while geometry file only generates a sample to show how the mesh would be embedded in the geometry.

B.3. SnappyHexMesh tool

This is one of the tools from OpenFOAM that can generate complex geometries. The process followed was the one specified in [27]. First, a *blockMesh* command was used to generate a cube-shaped mesh with bigger dimensions than the object in it. In this case, the T-junction mesh was the object. This file is shown in B.11, which is shorter and simpler than Figures B.1, B.2 and B.3.

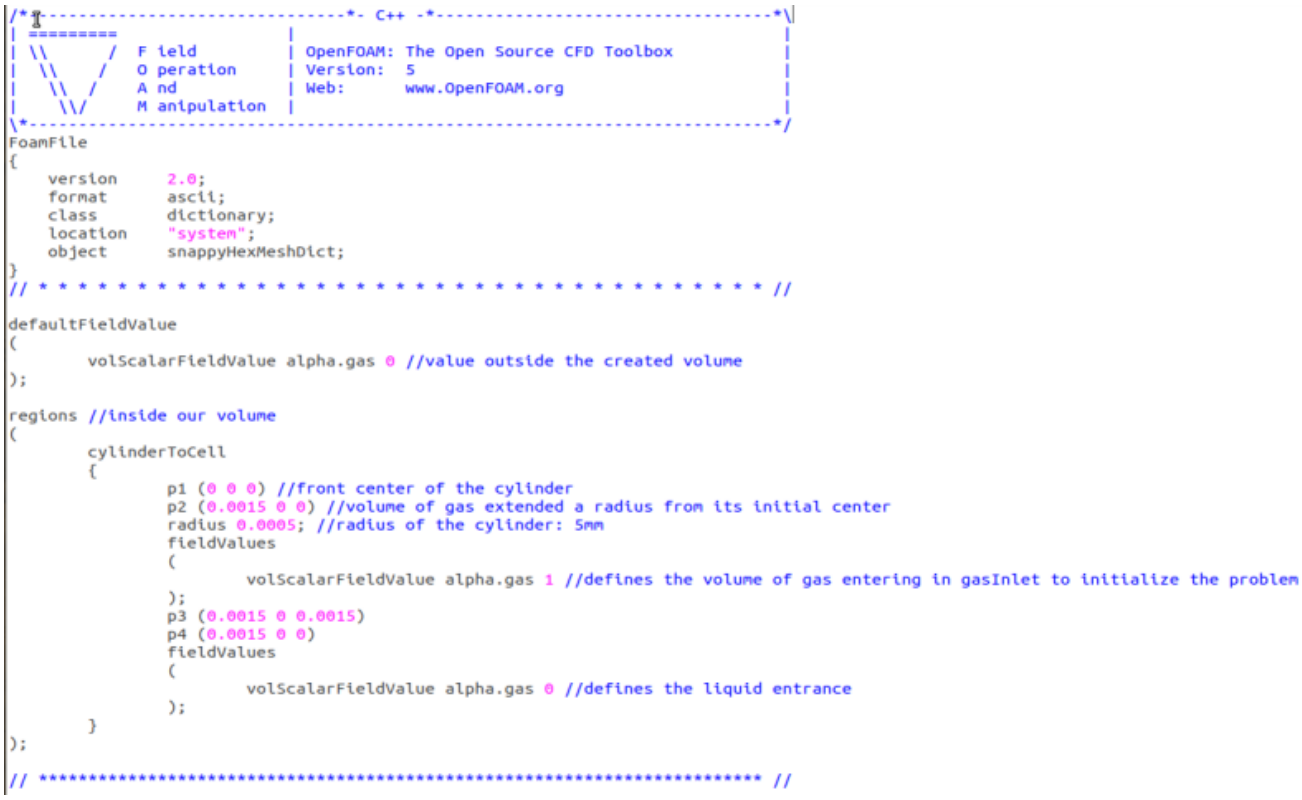
```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.001; //then I can write dimensions in mm
vertices
(
        //CUBE background mesh
        (0 -1 -1)       //0
        (0 1 -1)        //1
        (10 1 -1)       //2
        (10 -1 -1)      //3
        (0 -1 2)        //4
        (0 1 2)         //5
        (10 1 2)        //6
        (10 -1 2)       //7
);
blocks
(
        hex (0 1 5 4 3 2 6 7) (700 250 250) simpleGrading (1 1 1) //block 0
);
edges
(
);

boundary
(
        gasInlet
        {
                type patch;
                faces
                (
                        (0 1 5 4)
                );
        }

        liquidInlet
        {
                type patch;
                faces
                (
                        (4 5 6 7)
                );
        }
```

```
        outlet
        {
                type patch;
                faces
                (
                        (2 3 7 6)
                );
        }

        walls1
        {
                type wall;
                faces
                (
                        (0 4 7 3)
                        (1 2 6 5)
                );
        }

        walls2
        {
                type wall;
                faces
                (
                        (0 3 2 1)
                );
        }
);

mergePatchPairs
(
        //(walls1 mergewall)
        //(walls2 mergewall)
);

// ************************************************************************* //
```

Figure B.11: *BlockMeshDict* file for a cube-shaped mesh.

Besides the cube-shaped mesh, *snappyHexMesh* required a folder called ”geometry” in the simulation case to save the patches of the 3D geometry mesh. Exported face groups in ”.stl” format from www.onshape.com were saved in it, takind advantage of the website facilities.

With the help of the *snappyHexMeshDict* file, these STL patches can substract the undesired part of the cube-shaped mesh in ParaView ([27]). This process never got to an end because it came to my mind that the mesh from the outside could not be removed without external boundary patches that I did not have in the T-junction. Still, the aspect of the final *snappyHexMeshDict* file was is in Figure B.13, but the initial version from Figure B.12 was the base of it.



```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      snappyHexMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

defaultFieldValue
(
        volScalarFieldValue alpha.gas 0 //value outside the created volume
);

regions //inside our volume
(
        cylinderToCell
        {
                p1 (0 0 0) //front center of the cylinder
                p2 (0.0015 0 0) //volume of gas extended a radius from its initial center
                radius 0.0005; //radius of the cylinder: 5mm
                fieldValues
                (
                        volScalarFieldValue alpha.gas 1 //defines the volume of gas entering in gasInlet to initialize the problem
                );
                p3 (0.0015 0 0.0015)
                p4 (0.0015 0 0)
                fieldValues
                (
                        volScalarFieldValue alpha.gas 0 //defines the liquid entrance
                );
        }
);

// ************************************************************************* //
```

Figure B.12: First trial of *snappyHexMeshDict* file for a cylinder as a mesh.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      snappyHexMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

//which of these steps to run
castellatedMesh true;
snap            true;
addLayers       true;

mergeTolerance  1e-6;

//1. Define geometry
geometry
{
        mesh1 //.stl filename
        {
                type triSurfaceMesh;
                name mesh1;
                regions
                {
                        gasInlet
                        {
                                name    gasInlet;
                        }
                        liquidInlet
                        {
                                name    liquidInlet;
                        }
                        outlet
                        {
                                name    outlet;
                        }
                        walls1
                        {
                                name    walls1;
                        }
                        walls2
                        {
                                name    walls2;
                        }
                }
        }
        box1x1x1        //defined region name
        {
                type    searchableBox;  //region defined by bounding box
                min     (0 -0.5 -0.5);
```

```
                max     (10 0.5 1.5);
        }
}

//2.Refinement of the mesh
castellatedMeshControls
{
        maxGlobalCells          2000000;
        minRefinementCells      0;
        nCellsBetweenLevels     1;

        features
        (
                mesh1.stl
                {
                        file "mesh1.eMesh"; //file containing edge mesh
                        level 3; //level of refinement
                }
        );
        refinementSurfaces
        {
                gasInlet        {level(2 3); patchInfo { type patch; }}
                liquidInlet     {level(2 3); patchInfo { type patch; }}
                outlet          {level(3 4); patchInfo { type patch; }}
                walls1          {level(3 4); patchInfo { type wall; }}
                walls2          {level(3 4); patchInfo { type wall; }}
        }


        resolveFeatureAngle     30;
        refinementRegions
        {
                walls1
                {
                        mode inside;
                        levels((1E5 3));
                }
                walls2
                {
                        mode inside;
                        levels((1E15 3));
                }
                box1x1x1
                {
                        mode inside;
                        levels(2 3);
                }
                mesh1.stl
                {
                        mode distance;
                        levels((4 5) (2 3));
                }
        }
        locationInMesh  (0.009 0.00075 0.0015);
        allowFreeStandingZoneFaces      true;
}
```

```
snapControls
{
        nSmoothPatch            5;
        tolerance               1.0;
        nSolverIter             300;
        nRelaxIter              5;
        nFeatureSnapIter        10;
        implicitFeatureSnap     false;
        explicitFeatureSnap     true;
        multiregionFeatureSnap  true;
}
addLayersControls
{
        layers
        {
                gasInlet
                {
                        nSurfaceLayers  3;
                }
                liquidInlet
                {
                        nSurfaceLayers  3;
                }
                outlet
                {
                        nSurfaceLayers  4;
                }
                walls1
                {
                        nSurfaceLayers  4;
                }
                walls2
                {
                        nSurfaceLayers  4;
                }
        }
        relativeSizes                   true;
        expansionRatio                  1;
        minThickness                    0.1;

        //Advanced settings
        featureAngle                    50;
        maxFaceThicknessRatio           0.5;
        nSmoothSurfaceNormals           1;
        thickness                       1;
        nSmoothThickness                10;
        minMedialAxisAngle              90;
        maxThicknessToMedialRatio       0.3;
        nSmoothNormals                  3;
        nBufferCellsNoExtrude           0;
        nLayerIter                      10;
        nRelaxedIter                    20;
        nRelaxIter                      5;
}

 //********************************************************************** //
```

Figure B.13: *BlockMeshDict* file for a cube-shaped mesh.

B.4. OnShape website

This website (www.onshape.com) allows any user to create a free account and draw a geometry on-line without the need to download a CAD program. This way, geometry files do not occupy space on the local PC and they can also be exported from the website with any extension.

This website started being used because *snappyHexMesh* needed patches exported in an STL format, and OnShape could generate a geometry and export its face groups within less than five minutes. The website editing tools platform looks like a basic version of a CAD software and it is very intuitive.

When generating a geometry for other mesh generators like ANSYS, this website seemed like the fastest option. Programs like ANSYS could not select faces or parts of the geometry if this was exported in STL format, though. Every selection included the entire volume and that did not allow the face groups assignment. Thankfully OnShape also offered the possibility to export any geometry in STEP format, which fixed the problem.

Only two geometries were created in this website, both using international system units dimensions for compatibility reasons between programs. These appear as captions in section 2.3.2.1..

B.5. ANSYS

The last mesh generator used was ANSYS student package, which is a free software for college students from around the world. ANSYS is known to be very graphic and intuitive like CFD Autodesk software, which I used in previous subjects from college. ANSYS is installed in Windows, so the meshes were sent to Ubuntu through a shared directory between the local operating system and the virtual box of Ubuntu.

The quickest meshing in ANSYS is made through the Mesh generator of the program, which required the definition of a geometry and then the creation of the mesh according to it. The geometry of extension ".stp" was imported from OnShape. Once imported the geometry to ANSYS, the mesh itself was created. This one had to be made for "CFD" purposes in "Fluent" extension (".msh") to ease the conversion later on in OpenFOAM with the command *"fluent3DMeshToFoam nameMesh.msh"*. That way, ANSYS generated the modifiable mesh file in ".wpfj" extension and this one could also be exported as a MSH format.

The cell shape, the number of cells and their distribution were the parameters modified in every trial to obtain the best possible result.

As it is known from 2.3.2.1., the cylindrical T-junction did not work well enough for the same reason it did not either in GMSH, so the geometry ended up being changed. Although squared section or not, there are a lot of meshes generated that are around the virtual box in Ubuntu, each version improving the previous one in cell shape or size. The majority of them are shown in Figure B.14, excluding the ones used in the convergence tests and the two PDF documents.

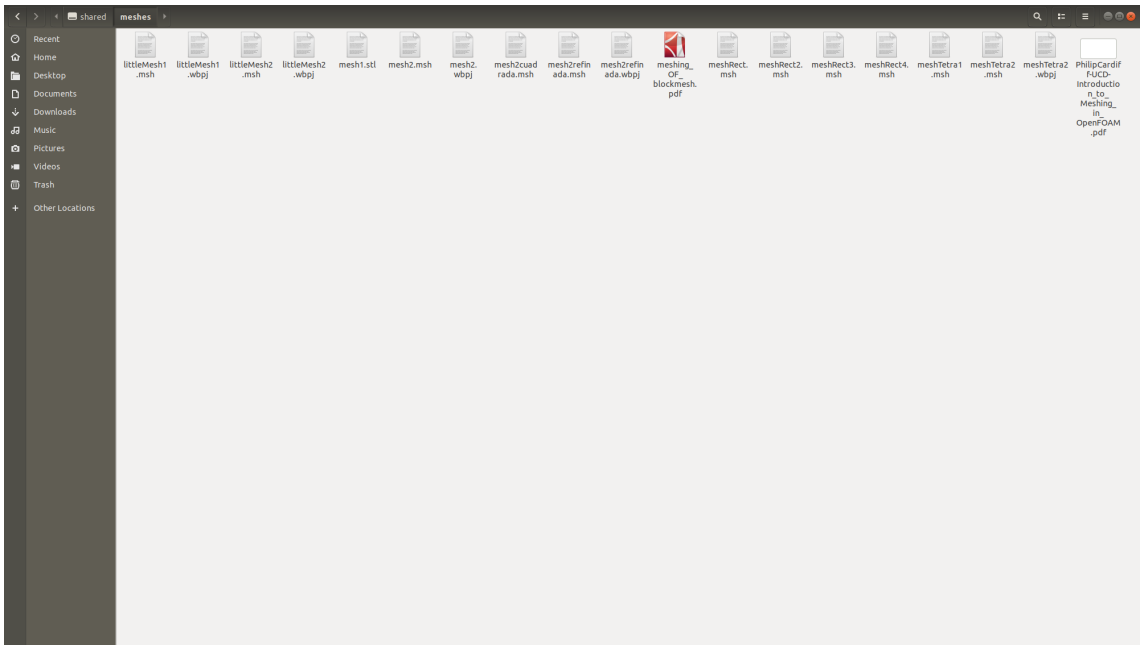


Figure B.14: Meshes imported to Ubuntu from the ANSYS software in Windows.

The rectangular mesh versions changed the number of cells to adjust the Courant number according to the parameters imposed in the simulation.

As for the cylindrical meshes, the initial shape and size fo the cells is shown in Figure B.15. The rest of them consisted on generating a layered mesh at the inlets and outlet and change the shape and size of the cells. Some examples of mesh modifications are shown in Figures B.16, B.17, B.18, B.19 and B.20. The combination of all these modifications brought more problems, though. In the end, my tutor and I came to an agreement to simplify the mesh as time was running out and there was still a lot of work to do.

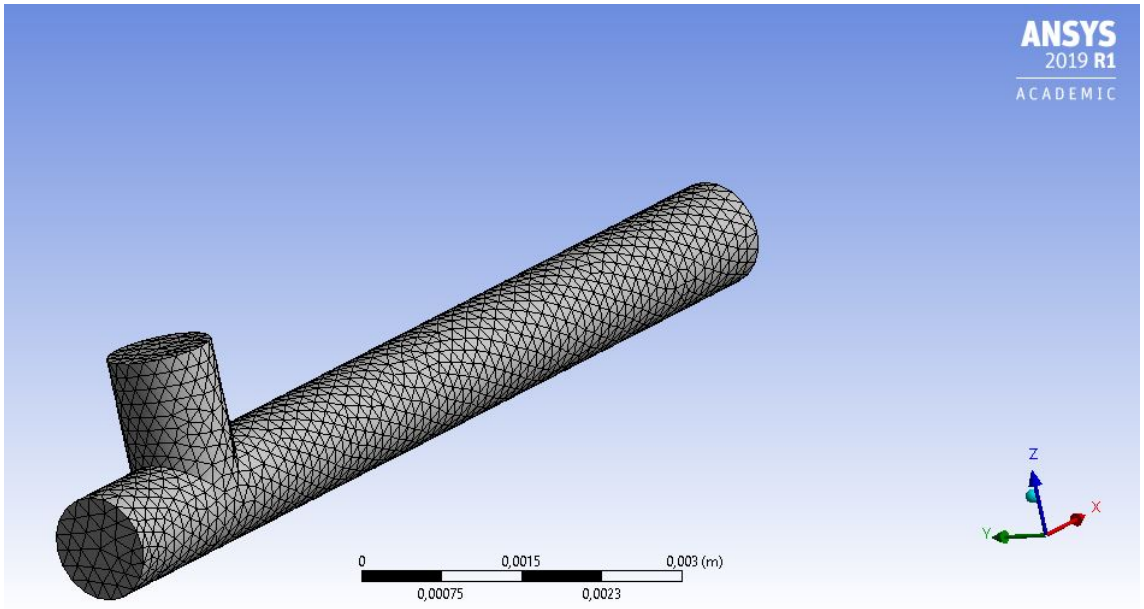


Figure B.15: First mesh generated in ANSYS with the default size and shape of the cells.

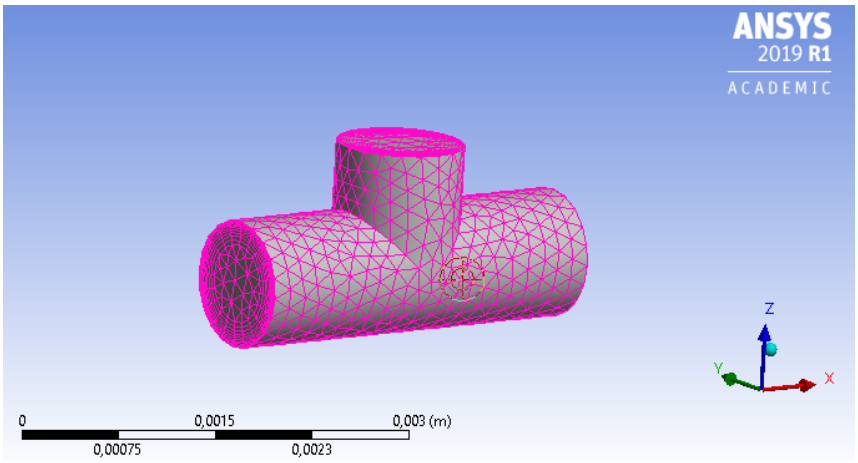


Figure B.16: First layered short mesh generated in ANSYS.

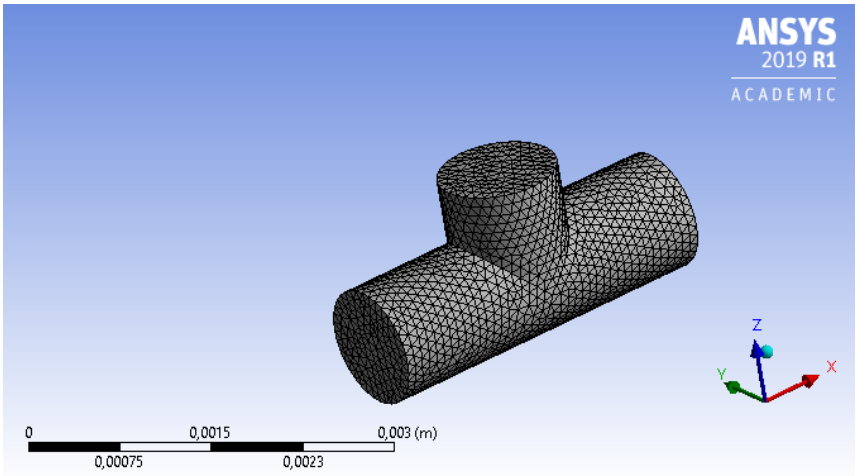


Figure B.17: Regular short mesh generated in ANSYS.

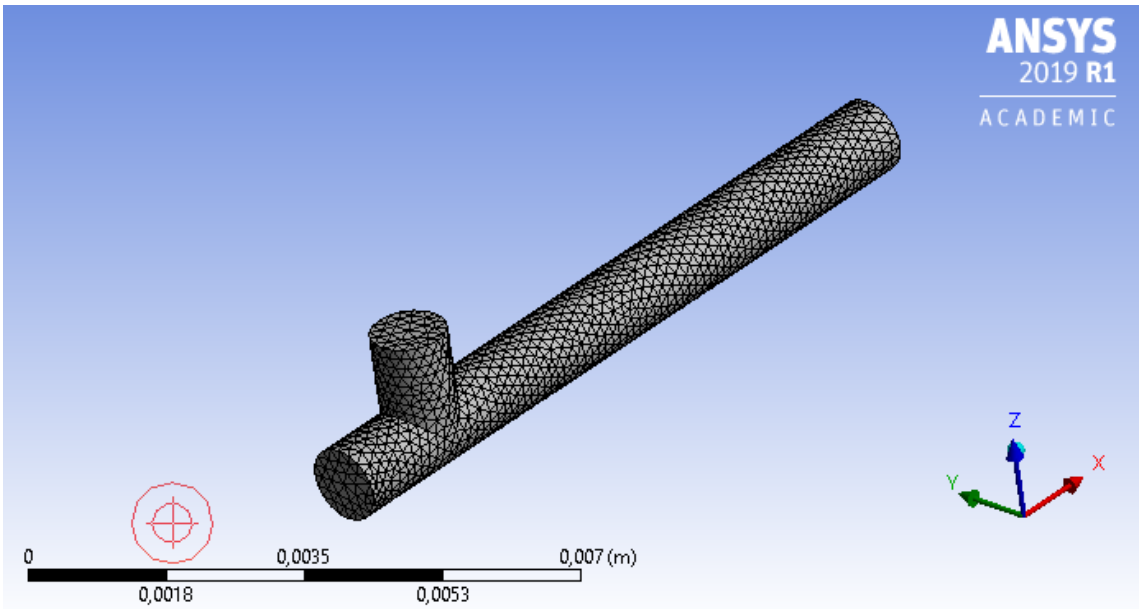


Figure B.18: Second mesh created in ANSYS.

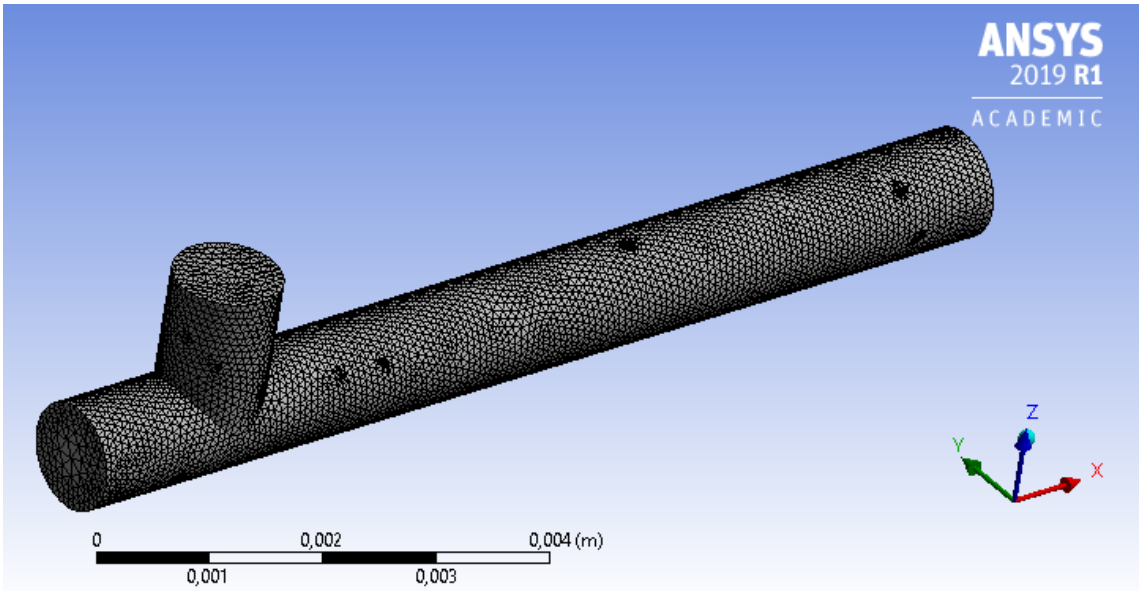


Figure B.19: Second mesh refined in ANSYS.

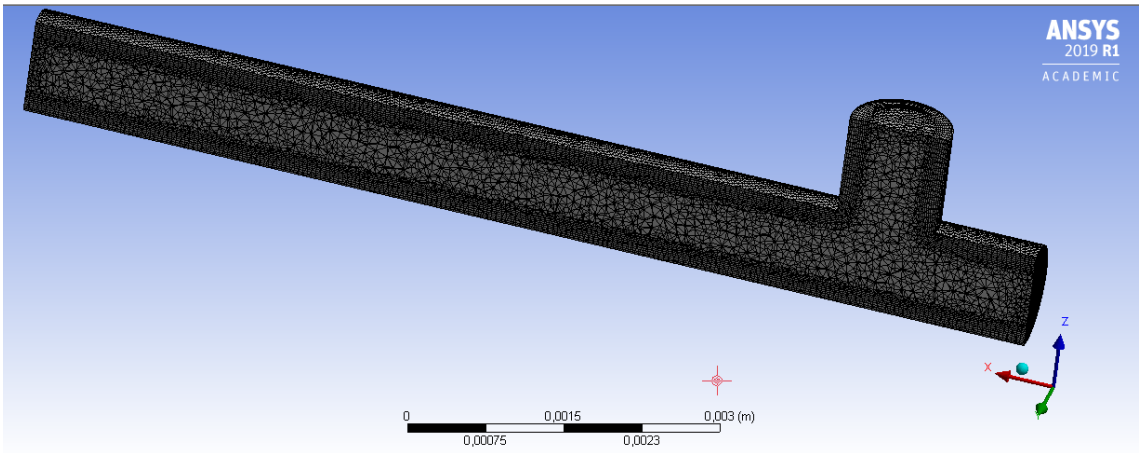


Figure B.20: Final layered mesh in ANSYS, half section.

The shortest meshes from above (Figures B.16 and B.17) were enough to evaluate the bubble detachment at the junction of the mesh and save time from computations. The ideal mesh should have layers so that the cell shape is not deformed at the junction between the two pipes. Still, both the layered case and the default cells distribution were simulated.

The same happened with the longer meshes. The second mesh (Figure B.18) was using a regular distribution of the cells, and when the size of the cells were modified (Figure B.19), some areas were more refined than others due to the cell shape and size. The change in geometry of the cell was possible but the concentrations of smaller cells would not disappear, so it made the Courant number rise. To minimize the effect of they caused in the computations, layers were added to improve the quality of the cells around the mesh (see Figure B.20). That did not work either, so that was the last trial before changing the geometry and break off the need to add layers to squared cells.

APPENDIX C. BOUNDARY CONDITIONS TRY OUTS

As explained in section 2.3.4., the chosen boundary conditions lead to some strange behaviour at the outlet of the mesh. This behaviour can be seen in the following sequence C.1.

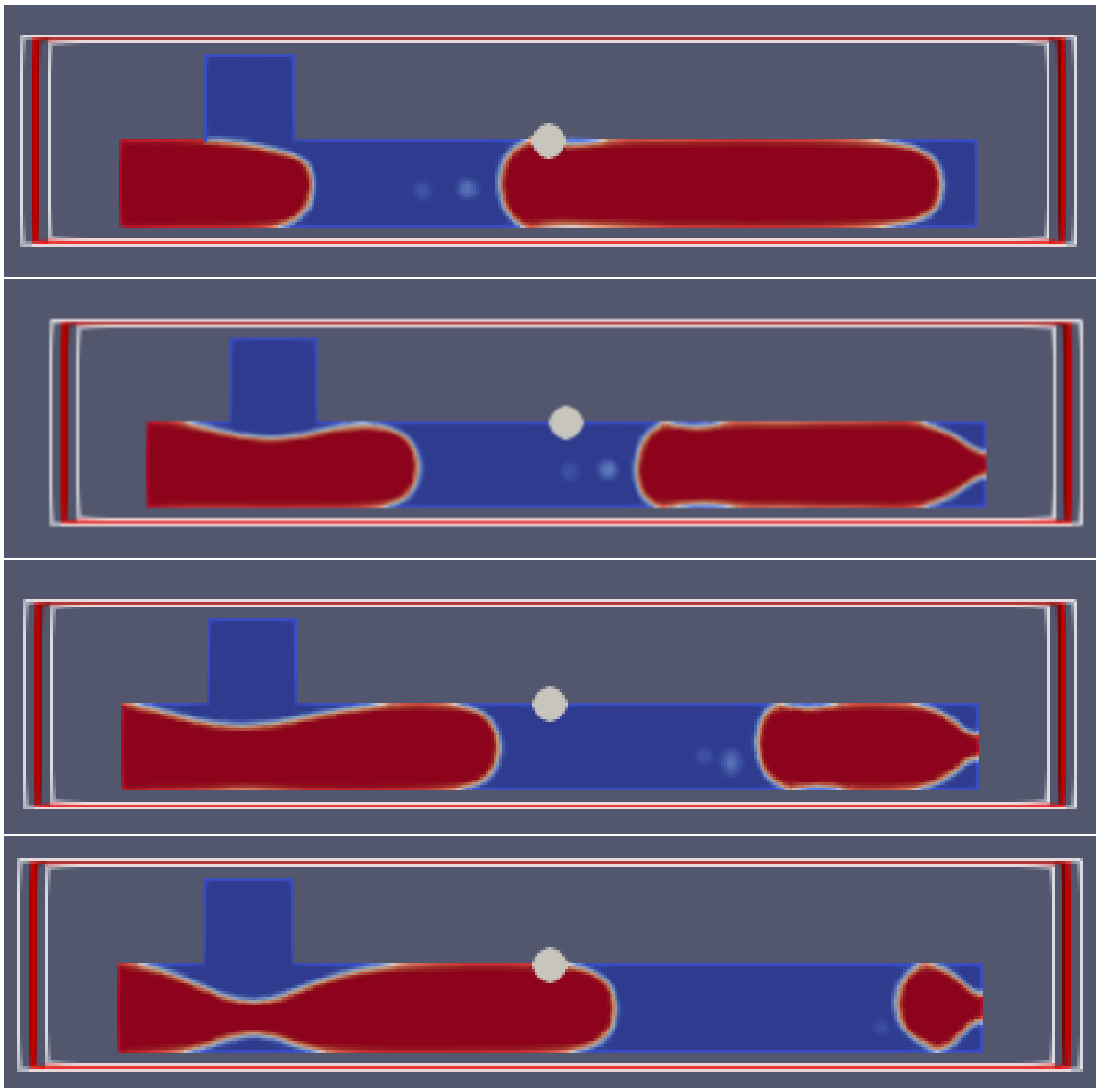


Figure C.1: Caption of the outflow sequence in ParaView visualizer.

From the previous sequence C.1, we conclude that the bubble travels through the pipe without any anomalies, but once it touches the outlet, some kind of vacuum process takes place and absorbs the bubble from the centre of the face. The chosen boundary conditions are from 2.3.4., but a lot of hours have been invested in trying to find better conditions for the outlet to improve the scenario. None of them improved it much though.

Then, some of the most recent boundary conditions changed will be posted in this section from now on. Not all of them though, as there are a lot of possibilities [29]. Most of them

were used at some point while testing. In the end, the boundary conditions from 2.3.4. were maintained, which are mostly coincident to the ones used in Blanca Dalfó and Amina Bakkali projects, [15] and [18].

At the beginning, when consulting tutorials from [22] and [23] and reading users experiences in www.cfd-online.com forums, it was confirmed that *fvSolution* and *fvSchemes* files were wrong. Apparently, a model of cavity simulation was used with different tolerances and numerical schemes that could have aggravated the behaviour. Once these were changed, some freeStream conditions from [29] were used for pressure and velocity, as it seemed these were the cause of the problem.

C.1. Change in the pressure condition

One of the most recent cases consisted on changing the outlet condition for the pressure and leave the rest as in section 2.3.4.. Figure C.2 shows this.

The idea was to define the absolute pressure instead of the gage one, but it turned out to be a condition for compressible flows, according to [30], which is not our case. The behaviour turned out to be even more strange at the outlet due to this condition, where the bubble exited the mesh through a smaller section of the outlet.

Back to the gage pressure, a similar behaviour was obtained when changing the outlet condition to a *freeStreamPressure* value of p0 = 0 Pa. Therefore, we thought the velocity at the walls and the outlet might have something to do with it. So maybe the solution was to change the velocity condition. After all, when visualizing the vectors of the velocity field in ParaView, these showed a bigger magnitude at the centre of the outlet than the rest of the face.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p_rgh;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 -1 -2 0 0 0 0];

internalField   uniform 101325; //initial state of the fluid, 0 means patm (pgage)

boundaryField
{
    gasinlet
    {
        type            zeroGradient;
    }

    liquidinlet
    {
        type            zeroGradient;
    }
    outlet
    {
        type            totalPressure;
        p0              uniform 101325; //Pabs
    }

    walls1
    {
        type            zeroGradient;
    }

    walls2
    {
        type            zeroGradient;
    }

}

// ************************************************************************* //
```

Figure C.2: Caption of the total pressure condition at the outlet.

C.2. Change in the velocity condition

Since the changes in pressure did not work as expected, the initial conditions for that field were left while the velocity outlet was changed in order to see if this was the wrong condition. At first, the walls got the ”noSlip” condition type, so these were changed to fix $0\,\mathrm{m/s}$ velocity for all axes. Nothing changed so far, so we tried other options, being one of them the following example.

The velocity condition of the outlet was changed in the ”U” file for an ”inletOutlet” boundary.

This condition computes the velocity value at the outlet (a *zeroGradient* with no restriction) but taking into account the inlet velocity. Since there are two inlets in the mesh, the inlet value was assigned as the sum of both inlet velocities, which were equal to $0.1\,\mathrm{m/s}$. Therefore, the inlet value was $0.2\,\mathrm{m/s}$.
The condition was a mistake for sure as there was no guarantee that the combined velocity would be exactly the sum and not some higher value. Matlab actuallythat this one was way higher in section x = 3 millim. However, the error produced would have been worth it if the scenario fixed itself with this condition.

It did not work. It seemed like there was something missing in this condition, like maybe the pressure and velocity conditions combined were the key to solve this.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0); //initial state of the fluid

boundaryField
{
    gasinlet
    {
        type            fixedValue;
        value           uniform (0.1 0 0);
    }

    liquidinlet
    {
        type            fixedValue;
        value           uniform (0 0 -0.1);
    }

    outlet
    {
        type            inletOutlet;
        inletValue      uniform (0.2 0 0); //Velocity at the joint is at least the sum of both inlet velocities
        value           $internalField;
    }

    walls1
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    walls2
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

}

// ************************************************************************* //
```

Figure C.3: Caption of the inletOutlet condition for the velocity at the outlet.

C.3. Change in both pressure and velocity conditions

The last example implies changes in both the pressure and the velocity fields. Changing the conditions for only one of the fields felt like it was not enough, so maybe the conditions imposed until now were not compatible and that made the simulation ”fail” somehow.

Several combinations were tried, especially using the *outletInlet* condition at the inlet for the pressure field and a *freeStream* velocity condition or so. Most of the examples for the outlet listed on [29] were combined.

There was one example that seemed to improve just a little the scenario but not enough to change the boundary conditions. With it, the bubbles seemed to increase its speed along the mesh and the outflow made the bubbles deform while crossing the outlet. In 2.3.4., the conditions maintained the round shape of it while exiting. After all, this combination did not make the vacuum effect disappear, so it was not better than what was written initially. The outflow sequence related to this change of conditions is shown in Figure C.4

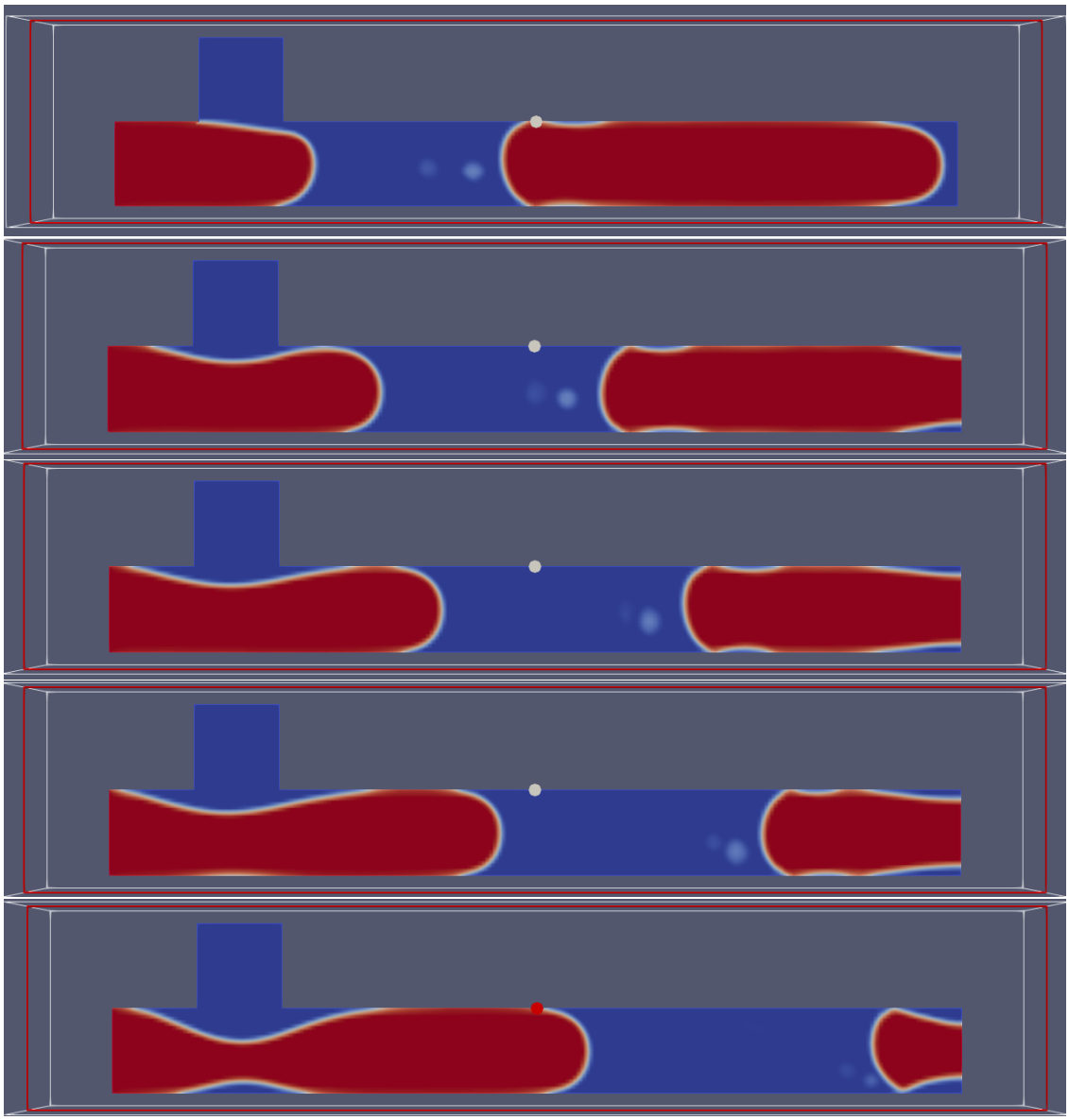


Figure C.4: Outflow sequence in ParaView for pressure and velocity change.

The conditions from the sequence C.4 has its files illustrated in pictures C.5 and C.6. Even when putting the pressure outlet condition in *gasInlet* instead, it did not improve either.

No combination of changes topped this one, so we thought to be best if the initial boundary conditions were put for the simulations as they were basic commands and made more sense due to the simplicity of the simulation physical problem.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p_rgh;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 -1 -2 0 0 0 0];

internalField   uniform 0; //initial state of the fluid, 0 means patm (pgage)

boundaryField
{
    gasinlet
    {
        type            zeroGradient;
    }

    liquidinlet
    {
        type            zeroGradient;
    }
    outlet
    {
        type            outletInlet;
        outletValue     uniform 0; //Pgage
    }

    walls1
    {
        type            zeroGradient;
    }

    walls2
    {
        type            zeroGradient;
    }

}

// ************************************************************************* //
```

Figure C.5: Caption of the outletInlet pressure condition at the outlet.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0); //initial state of the fluid

boundaryField
{
    gasinlet
    {
        type            fixedValue;
        value           uniform (0.1 0 0);
    }

    liquidinlet
    {
        type            fixedValue;
        value           uniform (0 0 -0.1);
    }

    outlet
    {
        type            inletOutlet;
        inletValue      uniform (0.2 0 0); //Sum of both inletsV
        value           $internalField;
    }

    walls1
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    walls2
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

}

// ************************************************************************* //
```

Figure C.6: Caption of the inletOutlet condition for the velocity at the outlet.

APPENDIX D. POST-PROCESSING FILES

This section of the appendix is related to the post processing files generated in each simulation. As it has been explained in 2.4.2., probes and sampled surfaces generate files with data from pressure and air fraction fields. These files are extremely long as there are thousands of time steps and points on each surface, so showing the first caption of each type of file is enough to know the pattern that the rest of the file follows.

D.1. Probes file

The pressure probe points are stored in the probes file. This one presents at its top the four points of measure in the mesh, which are numbered from 0 to 3 to identify these with one of the columns below. There is also a column for the time step of the simulation and its rows are aligned with the correspondent measures of gage pressure in each probe.

The initial stage of the file is in Figure D.1.

```
# Probe 0 (0.000501092 -0.0005 0.000499999)
# Probe 1 (0.0015 -0.0005 0.00123913)
# Probe 2 (0.00250682 -0.0005 0.0005)
# Probe 3 (0.00148303 -0.0005 0.00026087)
#       Probe             0             1             2             3
#        Time
        5e-06        240509        240538        207403        229550
        1e-05      -2679.13      -2387.56      -2152.77      -2411.88
      1.5e-05      -1662.74      -1476.07      -1337.97       -1495.7
        2e-05      -1459.99      -1293.97      -1173.29      -1311.69
      2.5e-05       -1282.2      -1134.48      -1029.32      -1150.42
        3e-05      -1089.02       -961.21      -873.654      -975.754
      3.5e-05        -925.2      -813.498      -741.121      -827.045
        4e-05      -771.273       -675.43      -617.393      -688.133
      4.5e-05      -402.419      -343.867      -321.786      -355.491
        5e-05      -204.912      -166.642        -163.6      -177.581
      5.5e-05      -89.1931      -62.5925      -70.6176      -73.0633
        6e-05      -11.8616       7.28789      -8.11341      -2.82792
      6.5e-05       20.8297       36.6954       18.3422       26.8422
        7e-05        60.039       71.2968       49.3569       61.6884
      7.5e-05       113.834       119.019       91.9991       109.658
        8e-05       163.332        163.19       131.439       154.034
      8.5e-05       199.161        195.13       159.975       186.144
        9e-05        221.62       215.333       178.059       206.494
      9.5e-05       238.202       229.983       191.185       221.264
       0.0001       253.408       243.383       203.187        234.78
     0.000105       266.461       254.822       213.436       246.322
      0.00011       277.298        264.23       221.871       255.827
     0.000115       294.683       279.618       235.616       271.313
      0.00012       306.458       290.023       244.926       281.808
     0.000125       336.766       317.116       269.063           309
      0.00013       356.046       334.323       284.417       326.313
     0.000135       375.958           352       300.179       344.084
      0.00014       376.882       352.516       300.679       344.662
     0.000145       376.468       351.895       300.161       344.089
      0.00015       372.178       347.716       296.489       339.972
     0.000155       341.473       319.775       271.673       312.026
      0.00016       355.828       332.563       283.089       324.899
     0.000165       358.424       334.396       284.757        326.79
      0.00017       350.891       327.419       278.578       319.843
     0.000175       363.779       338.705       288.646       331.201
      0.00018       359.752       334.793       285.185       327.324
     0.000185       344.238        320.48       272.489       313.035
      0.00019       333.054       310.093       263.286       302.679
     0.000195       284.029        265.63       223.782       258.185
       0.0002       179.657       171.312       139.967       163.784
     0.000205       98.2371       97.6816       74.5239       90.0429
      0.00021        157.17       150.504       121.558       143.035
     0.000215       137.215       132.362       105.455       124.888
      0.00022       118.299       115.104       90.1608       107.672
     0.000225       103.145       101.211        77.847       93.7927
      0.00023       84.7565       84.4453       62.9934        77.068
     0.000235       70.0018       71.0109       51.0922       63.6538
      0.00024       60.9857       62.6925       43.7451       55.3757
     0.000245       51.3873       53.8484       35.9275       46.5653
      0.00025       45.7146       48.5185       31.2385       41.2834
     0.000255       38.6682       42.0137       25.5012        34.817
      0.00026       35.0656       38.5406       22.4617       31.3914
     0.000265       26.3852       30.5831       15.4314       23.4681
```

Figure D.1: Caption of the beginning of probes file in the post-processing folder.

D.2. Sampled surfaces files

This section only shows two surface file types. Inside the post-processing folder there is another folder called ”surfacesamples”. Inside this one, there is a directory for every 50 time steps with a set of files in it (see Figure D.2). The files attached to this section are correspondent to the directory of 0.1 seconds of the simulation time.

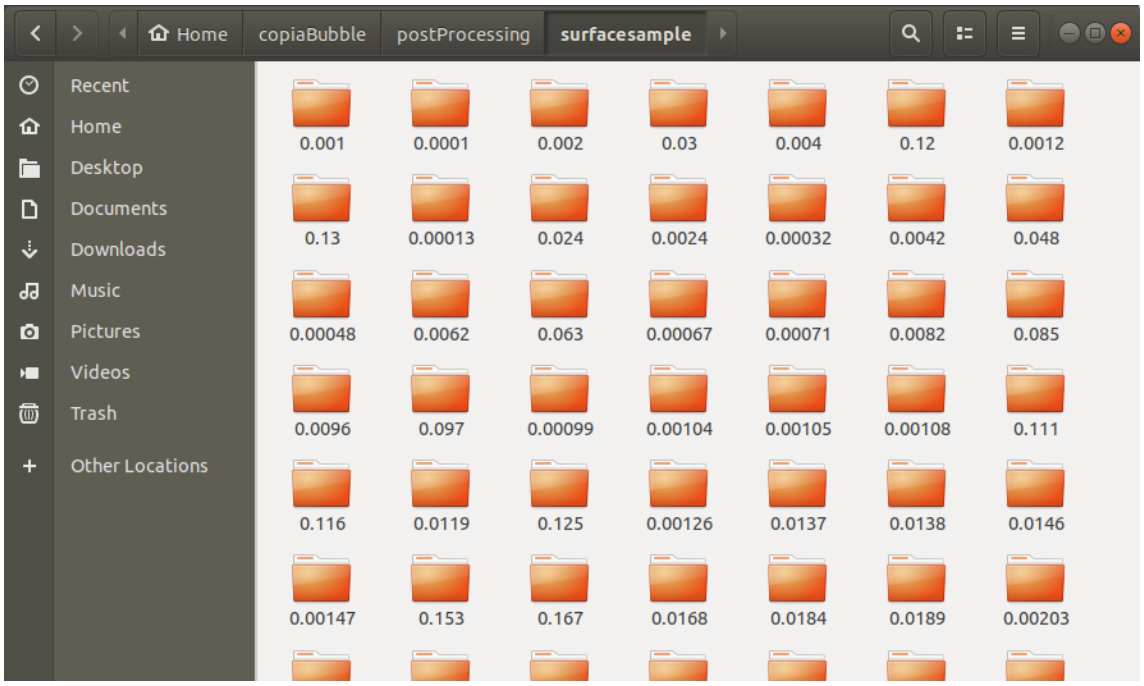


Figure D.2: Time directories inside the surface sampling folder, post-processing case.

Inside the directory of 0.1 seconds, there is a file for each surface: ”surface3mm”, ”surface4mm”, ”surface6mm”, ”surface7mm”, ”surface8mm”, ”surface9mm” and ”surfaceY”. The last one corresponds to the mesh mid plane of y = -0.0005 m, while the rest are related to their distance in the x-axis from the *gasInlet*.

All files are quite similar and follow the same pattern. However, the surfaces 9mm and Y are the ones shown in this section since they belong to different axes. In surface 9mm, all the points have the same x-coordinate (x = 0.009 m), as seen in Figure D.3. The other surfaces of the x-axis have x = 0.003 m constant or so, depending on their distance to the *gasInlet*. In the Y sampled surface, the x and the z coordinates are different in each line, but the y-coordinate is the one that stays the same as it is constant in the plane y = -0.0005 m (seen in Figure D.4).

Both files’ header indicated the field sampled in each file, the column distribution in each axis and the field value.

```
# alpha.gas  POINT_DATA 1469
#  x  y  z  alpha.gas
0.009 0 0 6.41896e-09
0.009 -4.34783e-06 0 5.70574e-09
0.009 0 4.35251e-06 5.70422e-09
0.009 -4.35248e-06 4.35251e-06 5.70422e-09
0.009 -2.17391e-05 2.17393e-05 1.14027e-08
0.009 0 3.90879e-05 3.7171e-08
0.009 -3.91304e-05 0 4.25875e-09
0.009 -4.39044e-06 3.90879e-05 3.7171e-08
0.009 -3.91258e-05 4.35251e-06 4.25761e-09
0.009 -4.34783e-05 0 5.0719e-09
0.009 0 4.34783e-05 4.84942e-08
0.009 -4.78261e-05 0 5.56073e-09
0.009 0 4.78781e-05 7.19509e-08
0.009 -4.78307e-05 4.35251e-06 6.86195e-09
0.009 -4.39984e-06 4.78781e-05 1.06593e-07
0.009 -4.34783e-05 4.34814e-05 5.75489e-08
0.009 -6.52174e-05 2.17393e-05 8.21608e-09
0.009 -2.17391e-05 6.52176e-05 2.06423e-07
0.009 0 8.25189e-05 6.92923e-08
0.009 -8.26087e-05 0 2.70707e-09
0.009 -4.43769e-06 8.25189e-05 1.04232e-07
0.009 -8.2604e-05 4.35251e-06 4.00905e-09
0.009 0 8.69565e-05 4.52717e-08
0.009 -8.69565e-05 0 1.86154e-09
0.009 -9.13043e-05 0 1.40509e-09
0.009 0 9.14037e-05 3.4174e-08
0.009 -9.1309e-05 4.35251e-06 1.40472e-09
0.009 -4.44719e-06 9.14037e-05 3.4174e-08
0.009 -6.52174e-05 6.52176e-05 7.50001e-09
0.009 -8.69565e-05 4.34814e-05 4.48578e-09
0.009 -4.34783e-05 8.69597e-05 5.32129e-08
0.009 -0.000108696 2.17393e-05 -7.74636e-15
0.009 -2.17391e-05 0.000108696 -4.33666e-15
0.009 -8.69565e-05 8.69597e-05 0.00329623
0.009 0 0.00012595 -7.59772e-14
0.009 -4.48494e-06 0.00012595 -7.59772e-14
0.009 -0.000126087 0 -1.32587e-13
0.009 -0.000126082 4.35251e-06 -1.32552e-13
0.009 -0.000108696 6.52176e-05 2.46351e-09
0.009 -6.52174e-05 0.000108696 2.23097e-09
0.009 -0.000130435 0 -1.6445e-13
0.009 0 0.000130435 -1.09034e-13
0.009 -0.000134783 0 -1.68373e-13
0.009 -0.000134787 4.35251e-06 -1.68376e-13
0.009 0 0.000134929 -1.23201e-13
0.009 -4.49454e-06 0.000134929 -1.23201e-13
0.009 -0.000130435 4.34814e-05 0.00229659
0.009 -4.34783e-05 0.000130438 0.00220755
0.009 -0.000152174 2.17393e-05 -2.93048e-13
0.009 -0.000108696 0.000108696 0.0131879
0.009 -2.17391e-05 0.000152174 -1.94825e-13
0.009 -0.000130435 8.69597e-05 0.0929058
0.009 -8.69565e-05 0.000130438 0.0929967
0.009 -0.000152174 6.52176e-05 0.0091903
0.009 -6.52174e-05 0.000152174 0.00883451
0.009 0 0.000169381 6.0634e-07
```

Figure D.3: Caption of the surface 9mm file in the post-processing folder for t = 0.1s.

```
# alpha.gas  POINT_DATA 12956
#  x  y  z  alpha.gas
0 -0.0005 -5.42101e-20 0.995338
0 -0.0005 2.17391e-05 1
2.18341e-05 -0.0005 -2.71051e-20 0.989791
2.1832e-05 -0.0005 2.17384e-05 0.99776
0 -0.0005 4.34783e-05 1
4.36681e-05 -0.0005 0 0.988553
4.36599e-05 -0.0005 4.34783e-05 0.998047
0 -0.0005 6.52174e-05 1
6.55022e-05 -0.0005 0 0.985804
2.18279e-05 -0.0005 6.52167e-05 1
6.5496e-05 -0.0005 2.17384e-05 0.9937
0 -0.0005 8.69565e-05 1
8.73362e-05 -0.0005 0 0.984266
6.54836e-05 -0.0005 6.52167e-05 1
4.36516e-05 -0.0005 8.69565e-05 1
8.73197e-05 -0.0005 4.34783e-05 0.995973
0 -0.0005 0.000108696 1
0.00010917 -0.0005 0 0.982232
2.18237e-05 -0.0005 0.000108695 1
0.00010916 -0.0005 2.17384e-05 0.989293
8.73032e-05 -0.0005 8.69565e-05 1
6.54712e-05 -0.0005 0.000108695 1
0.000109139 -0.0005 6.52167e-05 1
0 -0.0005 0.000130435 1
0.000131004 -0.0005 0 0.981276
4.36434e-05 -0.0005 0.000130435 1
0.00013098 -0.0005 4.34783e-05 0.99393
0 -0.0005 0.000152174 1
0.000152838 -0.0005 0 0.980947
2.18196e-05 -0.0005 0.000152173 1
0.000109119 -0.0005 0.000108695 1
0.000152824 -0.0005 2.17384e-05 0.986121
8.72867e-05 -0.0005 0.000130435 1
0.000130955 -0.0005 8.69565e-05 0.999971
6.54588e-05 -0.0005 0.000152173 1
0.000152795 -0.0005 6.52167e-05 0.999871
0 -0.0005 0.000173913 1
0.000174672 -0.0005 0 0.979875
4.36351e-05 -0.0005 0.000173913 1
0.000174639 -0.0005 4.34783e-05 0.992308
0.00013093 -0.0005 0.000130435 1
0.000109098 -0.0005 0.000152173 1
0.000152766 -0.0005 0.000108695 1
8.72702e-05 -0.0005 0.000173913 1
0.000174606 -0.0005 8.69565e-05 0.999855
0 -0.0005 0.000195652 1
0.000196507 -0.0005 0 0.979776
2.18155e-05 -0.0005 0.000195651 1
0.000196488 -0.0005 2.17384e-05 0.983715
6.54465e-05 -0.0005 0.000195651 1
0.000196451 -0.0005 6.52167e-05 0.999539
0.000152737 -0.0005 0.000152173 1
0 -0.0005 0.000217391 1
0.000130905 -0.0005 0.000173913 1
0.000174573 -0.0005 0.000130435 1
0.000218341 -0.0005 0 0.978651
4.36268e-05 -0.0005 0.000217391 1
```

Figure D.4: Caption of the surface Y file in the post-processing folder for t = 0.1s.

D.3. Average air fraction file per surface

Inside the post-processing folder there are other folders called ”surface3mean”, ”surface4mean”, ”surface6mean”, ”surface7mean”, ”surface8mean” and ”surface9mean”. These have a file in each that collects the main value of air fraction at every time step to compute the bubble parameters. The Y surface was not sampled in this case. It was only used to draw a contour plot in Matlab.

As all these files are quite similar, just showing the file of the first sampled surface (x = 0.003 m in Figure D.5) is enough to see what the file is about.

```
# Region type : sampledSurface surface3mean
# Faces       : 2520
# Area        : 1.000000e-06
# Scale factor: 1.000000e+00
# Time          average(alpha.gas)
5e-06           7.728640e-128
1e-05           8.223877e-128
1.5e-05         8.733243e-128
2e-05           9.261920e-128
2.5e-05         9.811273e-128
3e-05           1.038267e-127
3.5e-05         1.097773e-127
4e-05           1.159804e-127
4.5e-05         1.224533e-127
5e-05           1.292291e-127
5.5e-05         1.363336e-127
6e-05           1.437897e-127
6.5e-05         1.516200e-127
7e-05           1.598449e-127
7.5e-05         1.684872e-127
8e-05           1.775723e-127
8.5e-05         1.871274e-127
9e-05           1.971805e-127
9.5e-05         2.077602e-127
0.0001          2.188968e-127
0.000105        2.306219e-127
0.00011         2.429694e-127
0.000115        2.559750e-127
0.00012         2.696773e-127
0.000125        2.841173e-127
0.00013         2.993413e-127
0.000135        3.153971e-127
0.00014         3.323364e-127
0.000145        3.502113e-127
0.00015         3.690769e-127
0.000155        3.889912e-127
0.00016         4.100102e-127
0.000165        4.322024e-127
0.00017         4.556382e-127
0.000175        4.803908e-127
0.00018         5.065425e-127
0.000185        5.341771e-127
0.00019         5.633806e-127
0.000195        5.942448e-127
0.0002          6.268550e-127
0.000205        6.612808e-127
0.00021         6.975986e-127
0.000215        7.359382e-127
0.00022         7.764088e-127
0.000225        8.191259e-127
0.00023         8.642121e-127
0.000235        9.117945e-127
0.00024         9.620087e-127
0.000245        1.015000e-126
0.00025         1.070920e-126
0.000255        1.129932e-126
0.00026         1.192207e-126
0.000265        1.257928e-126
0.00027         1.327283e-126
```

Figure D.5: Caption of the surface 3 file for average air fraction values.

APPENDIX E. MATLAB CODES

The codes used for this project are made from scratch, but following the same criteria as Blanca Dalfó and Amina Bakkali projects [[15] and [18]].

E.1. Surface samplings and contour plots

First of all, most of the simulations went through this code to save the sampled data in Matlab ”structs” and organize this information for future consultations in other codes. The main code also includes the possibility to draw the contour of a sampled surface, as well as plotting the evolution of the air fraction per time, whether it is through the sampled surface mid-point or using mean values.

The last section of the main code obtains the area and the initial and final times of bubbles crossing each surface. This information is stored in new ”structs” to import them to parameters computation codes. These are lighter, so they run the codes quicker.

```
%BUBBLE PLOTS FROM SAMPLED SURFACES
%Run this section for each mesh: Lighter mesh (L), Medium mesh (M),
%Denser mesh (D).
clear all
format long; casesList = sort(string(strsplit(ls("/home/gis/bubbleFV11/
  postProcessing/surfacesample/"))));
casesList(1) = [];
%I create a list of directories
%Creating the data frame
s6 = struct("time", [], "data",struct("x", [], "y", [], "z", [], "alpha", [],
   "dy", [], "dz", []), "midPoint", struct("y", 0, "z", 0), "alphaMidP", [],
  "meanAlpha", [], "realTime", []);
s3 = s6;
s4 = s6;
s7 = s6;
s8 = s6;
s9 = s6;

%Surfaces alpha plots
for i=1:1:length(casesList)
%Alpha per each surface point
[f3, err3] = fopen("/home/gis/bubbleFV11/postProcessing/surfacesample/"+
  casesList(i)+"/alpha.gas_surface3mm.raw", 'r');
[f4, err4] = fopen("/home/gis/bubbleFV11/postProcessing/surfacesample/"+
  casesList(i)+"/alpha.gas_surface4mm.raw", 'r');
[f6, err6] = fopen("/home/gis/bubbleFV11/postProcessing/surfacesample/"+
  casesList(i)+"/alpha.gas_surface6mm.raw", 'r');
[f7, err7] = fopen("/home/gis/bubbleFV11/postProcessing/surfacesample/"+
  casesList(i)+"/alpha.gas_surface7mm.raw", 'r');
[f8, err8] = fopen("/home/gis/bubbleFV11/postProcessing/surfacesample/"+
  casesList(i)+"/alpha.gas_surface8mm.raw", 'r');
[f9, err9] = fopen("/home/gis/bubbleFV11/postProcessing/surfacesample/"+
  casesList(i)+"/alpha.gas_surface9mm.raw", 'r');
%Mean Alpha per surface
[f3m, err3m] = fopen("/home/gis/bubbleFV11/postProcessing/surface3mean/0/
  surfaceFieldValue.dat", 'r');
```

```
d = textscan(f3m, "%f64 %f64", "HeaderLines", 5, "Delimiter", "\n");
fclose(f3m);
d3 = cell2mat(d);
s3.realTime = d3(:,1);
s3.meanAlpha = d3(:,2);
[f4m, err4m] = fopen("/home/gis/bubbleFV11/postProcessing/surface4mean/0/
  surfaceFieldValue.dat", 'r');
d = textscan(f4m, "%f64 %f64", "HeaderLines", 5, "Delimiter", "\n");
fclose(f4m);
d4 = cell2mat(d);
s4.realTime = d4(:,1);
s4.meanAlpha = d4(:,2);
[f6m, err6m] = fopen("/home/gis/bubbleFV11/postProcessing/surface6mean/0/
  surfaceFieldValue.dat", 'r');
d = textscan(f6m, "%f64 %f64", "HeaderLines", 5, "Delimiter", "\n");
fclose(f6m);
d6 = cell2mat(d);
s6.realTime = d6(:,1);
s6.meanAlpha = d6(:,2);
[f7m, err7m] = fopen("/home/gis/bubbleFV11/postProcessing/surface7mean/0/
  surfaceFieldValue.dat", 'r');
d = textscan(f7m, "%f64 %f64", "HeaderLines", 5, "Delimiter", "\n");
fclose(f7m);
d7 = cell2mat(d);
s7.realTime = d7(:,1);
s7.meanAlpha = d7(:,2);
[f8m, err8m] = fopen("/home/gis/bubbleFV11/postProcessing/surface8mean/0/
  surfaceFieldValue.dat", 'r');
d = textscan(f8m, "%f64 %f64", "HeaderLines", 5, "Delimiter", "\n");
fclose(f8m);
d8 = cell2mat(d);
s8.realTime = d8(:,1);
s8.meanAlpha = d8(:,2);
[f9m, err9m] = fopen("/home/gis/bubbleFV11/postProcessing/surface9mean/0/
  surfaceFieldValue.dat", 'r');
d = textscan(f9m, "%f64 %f64", "HeaderLines", 5, "Delimiter", "\n");
fclose(f9m);
d9 = cell2mat(d);
s9.realTime = d9(:,1);
s9.meanAlpha = d9(:,2);
if f3 == -1
  fprintf("========\nCannot open the file f3: error %g, %g\n", err3, i);
elseif f4 == -1
    fprintf("========\nCannot open the file f4: error %g, %g\n", err4, i);
elseif f6 == -1
    fprintf("========\nCannot open the file f6: error %g, %g\n", err6, i);
elseif f7 == -1
    fprintf("========\nCannot open the file f7: error %g, %g\n", err7, i);
elseif f8 == -1
    fprintf("========\nCannot open the file f8: error %g, %g\n", err8, i);
elseif f9 == -1
    fprintf("========\nCannot open the file f9: error %g, %g\n", err9, i);
elseif f3m == -1
    fprintf("========\nCannot open the file f3m: error %g, %g\n", err3m, i);
elseif f4m == -1
    fprintf("========\nCannot open the file f4m: error %g, %g\n", err4m, i);
elseif f6m == -1
    fprintf("========\nCannot open the file f6m: error %g, %g\n", err6m, i);
```

```
elseif f7m == -1
    fprintf('========\nCannot open the file f7m: error %g, %g\n', err7m, i);
elseif f8m == -1
    fprintf('========\nCannot open the file f8m: error %g, %g\n', err8m, i);
elseif f9m == -1
    fprintf('========\nCannot open the file f3m: error %g, %g\n', err9m, i);
else
  format long;
    s3.time(i)= double(string(casesList(i)));
    s4.time(i)= double(string(casesList(i)));
    s6.time(i)= double(string(casesList(i)));
  s7.time(i)= double(string(casesList(i)));
  s8.time(i)= double(string(casesList(i)));
    s9.time(i)= double(string(casesList(i)));
%Fill s3
    n = 1;
    while ~feof(f3)
      line = fgetl(f3);
      words = strsplit(line, ' ');
      if strcmp(words(1),'#') == 0 % 0 to be different, 1 to be equal
       %First value is at the top of the file. Many values for each time(i) -
  data(i)
        format long;
        s3.data(i).x(n) = double(string(words(1)));
        s3.data(i).y(n) = double(string(words(2)));
        s3.data(i).z(n) = double(string(words(3)));
        s3.data(i).alpha(n) = double(string(words(4)));
        n = n+1;
      end
    end
    fclose(f3);
%Fill s4
    n = 1;
    while ~feof(f4)
      line = fgetl(f4);
        words = strsplit(line, ' ');
        if strcmp(words(1),'#') == 0
          format long;
          s4.data(i).x(n) = double(string(words(1)));
          s4.data(i).y(n) = double(string(words(2)));
          s4.data(i).z(n) = double(string(words(3)));
          s4.data(i).alpha(n) = double(string(words(4)));
          n = n+1;
        end
    end
      fclose(f4);
  %Fill s6
      n = 1;
      while ~feof(f6)
        line = fgetl(f6);
        words = strsplit(line, ' ');
        if strcmp(words(1),'#') == 0
          format long;
            s6.data(i).x(n) = double(string(words(1)));
            s6.data(i).y(n) = double(string(words(2)));
            s6.data(i).z(n) = double(string(words(3)));
              s6.data(i).alpha(n) = double(string(words(4)));
              n = n+1;
```

```
            end
        end
          fclose(f6);
%Fill s7
    n = 1;
    while ~feof(f7)
      line = fgetl(f7);
      words = strsplit(line, ' ');
      if strcmp(words(1),'#') == 0
        format long;
        s7.data(i).x(n) = double(string(words(1)));
        s7.data(i).y(n) = double(string(words(2)));
        s7.data(i).z(n) = double(string(words(3)));
        s7.data(i).alpha(n) = double(string(words(4)));
        n = n+1;
      end
    end
    fclose(f7);
%Fill s8
    n = 1;
    while ~feof(f8)
      line = fgetl(f8);
      words = strsplit(line, ' ');
      if strcmp(words(1),'#') == 0
        format long;
        s8.data(i).x(n) = double(string(words(1)));
        s8.data(i).y(n) = double(string(words(2)));
        s8.data(i).z(n) = double(string(words(3)));
        s8.data(i).alpha(n) = double(string(words(4)));
        n = n+1;
      end
    end
    fclose(f8);
%Fill s9
        n = 1;
        while ~feof(f9)
          line = fgetl(f9);
          words = strsplit(line, ' ');
          if strcmp(words(1),'#') == 0
            format long;
              s9.data(i).x(n) = double(string(words(1)));
              s9.data(i).y(n) = double(string(words(2)));
              s9.data(i).z(n) = double(string(words(3)));
              s9.data(i).alpha(n) = double(string(words(4)));
              n = n+1;
          end
    end
      fclose(f9);
end

%Find midPoint alpha, s3
for t=1:1:length(s3.time)
    for p=1:1:length(s3.data(t).y)
        format long;
        s3.data(t).dy(p) = abs(s3.data(t).y(p)+0.0005); %y<0
        s3.data(t).dz(p) = abs(s3.data(t).z(p)-0.0005); %z>0
    end
    %Let's find the minimum difference with the midPoint, which should have a
```

```
  common index for both y and z.
    minimaY = min(s3.data(t).dy);
    [miny, indexy] = find(s3.data(t).dy == minimaY);
    for q = 1:1:length(indexy)
      minzV(q) = s3.data(t).dz(indexy(q));
  end
    minimaZ = min(minzV);
    [minz, indexz] = find(minzV == minimaZ);
    zReal = indexy(indexz);
    format long;
    s3.midPoint(t).y = s3.data(t).y(zReal);
    s3.midPoint(t).z = s3.data(t).z(zReal);
    s3.alphaMidP(t) = s3.data(t).alpha(zReal);
end
timecontour = input("Insert a number from 1 to 1000 (time increases): ");
[meshCy, meshCz] = meshgrid(unique(s3.data(timecontour).y), unique(s3.data(
  timecontour).z));
F = scatteredInterpolant(s3.data(timecontour).y', s3.data(timecontour).z', s3
  .data(timecontour).alpha');
interpolated_data = F(meshCy, meshCz);
figure(1)
surf(meshCy, meshCz, interpolated_data,'EdgeColor', 'none');
title("Contour plot of S3mm alpha, seen from gasInlet"); xlabel("y axis");
  ylabel("z axis");
axis([-0.001 0 0 0.001 0 1])
view(-180,-90) %changes the view of the plane, as if it was seen from the
  gasInlet (-Y)
colormap jet; caxis([0 1]); colorbar

%Find midPoint alpha, s4
for t=1:1:length(s4.time)
  for p=1:1:length(s4.data(t).y)
      format long;
      s4.data(t).dy(p) = abs(s4.data(t).y(p)+0.0005); %y<0
      s4.data(t).dz(p) = abs(s4.data(t).z(p)-0.0005); %z>0
  end
  minimaY = min(s4.data(t).dy);
    [miny, indexy] = find(s4.data(t).dy == minimaY);
    for q = 1:1:length(indexy)
      minzV(q) = s4.data(t).dz(indexy(q));
    end
  minimaZ = min(minzV);
    [minz, indexz] = find(minzV == minimaZ);
    zReal = indexy(indexz);
    format long;
    s4.midPoint(t).y = s4.data(t).y(zReal);
    s4.midPoint(t).z = s4.data(t).z(zReal);
    s4.alphaMidP(t) = s4.data(t).alpha(zReal);
end
timecontour = input("Insert a number from 1 to 1000 (time increases): ");
[meshCy, meshCz] = meshgrid(unique(s4.data(timecontour).y), unique(s4.data(
  timecontour).z));
F = scatteredInterpolant(s4.data(timecontour).y', s4.data(timecontour).z', s4
  .data(timecontour).alpha');
interpolated_data = F(meshCy, meshCz);
figure(2)
surf(meshCy, meshCz, interpolated_data,'EdgeColor', 'none');
title("Contour plot of S4mm alpha, seen from gasInlet"); xlabel("y axis");
```

```
    ylabel("z axis");
246 axis([-0.001 0 0 0.001 0 1])
247 view(-180,-90) %changes the view of the plane, as if it was seen from the
    gasInlet (-Y)
248 colormap jet; caxis([0 1]); colorbar
249
250 %Find midPoint alpha, s6
251 for t=1:1:length(s6.time)
252   for p=1:1:length(s6.data(t).y)
253         format long;
254         s6.data(t).dy(p) = abs(s6.data(t).y(p)+0.0005); %y<0
255         s6.data(t).dz(p) = abs(s6.data(t).z(p)-0.0005); %z>0
256     end
257     minimaY = min(s6.data(t).dy);
258     [miny, indexy] = find(s6.data(t).dy == minimaY);
259     for q = 1:1:length(indexy)
260       minzV(q) = s6.data(t).dz(indexy(q));
261   end
262     minimaZ = min(minzV);
263   [minz, indexz] = find(minzV == minimaZ);
264     zReal = indexy(indexz);
265     format long;
266     s6.midPoint(t).y = s6.data(t).y(zReal);
267     s6.midPoint(t).z = s6.data(t).z(zReal);
268     s6.alphaMidP(t) = s6.data(t).alpha(zReal);
269 end
270 timecontour = input("Insert a number from 1 to 1000 (time increases): ");
271 [meshCy, meshCz] = meshgrid(unique(s6.data(timecontour).y), unique(s6.data(
    timecontour).z));
272 F = scatteredInterpolant(s6.data(timecontour).y', s6.data(timecontour).z', s6
    .data(timecontour).alpha');
273 interpolated_data = F(meshCy, meshCz);
274 figure(3)
275 surf(meshCy, meshCz, interpolated_data,'EdgeColor', 'none');
276 title("Contour plot of S6mm alpha, seen from gasInlet"); xlabel("y axis");
    ylabel("z axis");
277 axis([-0.001 0 0 0.001 0 1])
278 view(-180,-90) %changes the view of the plane, as if it was seen from the
    gasInlet (-Y)
279 colormap jet; caxis([0 1]); colorbar
280
281 %Find midPoint alpha, s7
282 for t=1:1:length(s7.time)
283   for p=1:1:length(s7.data(t).y)
284     format long;
285     s7.data(t).dy(p) = abs(s7.data(t).y(p)+0.0005); %y<0
286     s7.data(t).dz(p) = abs(s7.data(t).z(p)-0.0005); %z>0
287   end
288   minimaY = min(s7.data(t).dy);
289   [miny, indexy] = find(s7.data(t).dy == minimaY);
290   for q = 1:1:length(indexy)
291     minzV(q) = s7.data(t).dz(indexy(q));
292   end
293   minimaZ = min(minzV);
294   [minz, indexz] = find(minzV == minimaZ);
295   zReal = indexy(indexz);
296   format long;
297   s7.midPoint(t).y = s7.data(t).y(zReal);
```

```
    s7.midPoint(t).z = s7.data(t).z(zReal);
    s7.alphaMidP(t) = s7.data(t).alpha(zReal);
  end
  timecontour = input("Insert a number from 1 to 1000 (time increases): ");
  [meshCy, meshCz] = meshgrid(unique(s7.data(timecontour).y), unique(s7.data(
    timecontour).z));
  F = scatteredInterpolant(s7.data(timecontour).y', s7.data(timecontour).z', s7
    .data(timecontour).alpha');
  interpolated_data = F(meshCy, meshCz);
  figure(4)
  surf(meshCy, meshCz, interpolated_data,'EdgeColor', 'none');
  title("Contour plot of S7mm alpha, seen from gasInlet"); xlabel("y axis");
    ylabel("z axis");
  axis([-0.001 0 0 0.001 0 1])
  view(-180,-90) %changes the view of the plane, as if it was seen from the
    gasInlet (-Y)
  colormap jet; caxis([0 1]); colorbar

  %Find midPoint alpha, s8
  for t=1:1:length(s8.time)
    for p=1:1:length(s8.data(t).y)
      format long;
      s8.data(t).dy(p) = abs(s8.data(t).y(p)+0.0005); %y<0
      s8.data(t).dz(p) = abs(s8.data(t).z(p)-0.0005); %z>0
    end
    minimaY = min(s8.data(t).dy);
    [miny, indexy] = find(s8.data(t).dy == minimaY);
    for q = 1:1:length(indexy)
      minzV(q) = s8.data(t).dz(indexy(q));
    end
    minimaZ = min(minzV);
    [minz, indexz] = find(minzV == minimaZ);
    zReal = indexy(indexz);
    format long;
    s8.midPoint(t).y = s8.data(t).y(zReal);
    s8.midPoint(t).z = s8.data(t).z(zReal);
    s8.alphaMidP(t) = s8.data(t).alpha(zReal);
  end
  timecontour = 605; %input("Insert a number from 1 to 1000 (time increases):
    ");
  [meshCy, meshCz] = meshgrid(unique(s8.data(timecontour).y), unique(s8.data(
    timecontour).z));
  F = scatteredInterpolant(s8.data(timecontour).y', s8.data(timecontour).z', s8
    .data(timecontour).alpha');
  interpolated_data = F(meshCy, meshCz);
  figure(5)
  surf(meshCy, meshCz, interpolated_data,'EdgeColor', 'none');
  title("Contour plot of S8mm alpha, seen from gasInlet", "Fontsize",15);
    xlabel("y axis","Fontsize",12); ylabel("z axis", "Fontsize",12);
  axis([-0.001 0 0 0.001 0 1])
  view(-180,-90) %changes the view of the plane, as if it was seen from the
    gasInlet (-Y)
  colormap jet; caxis([0 1]); colorbar

  %Find midPoint alpha, s9
  for t=1:1:length(s9.time)
    for p=1:1:length(s9.data(t).y)
        format long;
```

```
347             s9.data(t).dy(p) = abs(s9.data(t).y(p)+0.0005); %y<0
348             s9.data(t).dz(p) = abs(s9.data(t).z(p)-0.0005); %z>0
349         end
350         minimaY = min(s9.data(t).dy);
351         [miny, indexy] = find(s9.data(t).dy == minimaY);
352         for q = 1:1:length(indexy)
353           minzV(q) = s9.data(t).dz(indexy(q));
354         end
355         minimaZ = min(minzV);
356         [minz, indexz] = find(minzV == minimaZ);
357         zReal = indexy(indexz);
358         format long;
359         s9.midPoint(t).y = s9.data(t).y(zReal);
360         s9.midPoint(t).z = s9.data(t).z(zReal);
361         s9.alphaMidP(t) = s9.data(t).alpha(zReal);
362     end
363     timecontour = input("Insert a number from 1 to 1000 (time increases): ");
364     [meshCy, meshCz] = meshgrid(unique(s9.data(timecontour).y), unique(s9.data(
        timecontour).z));
365     F = scatteredInterpolant(s9.data(timecontour).y', s9.data(timecontour).z', s9
        .data(timecontour).alpha');
366     interpolated_data = F(meshCy, meshCz);
367     figure(6)
368     surf(meshCy, meshCz, interpolated_data,'EdgeColor', 'none');
369     title("Contour plot of S9mm alpha, seen from gasInlet"); xlabel("y axis");
        ylabel("z axis");
370     axis([-0.001 0 0 0.001 0 1])
371     view(-180,-90) %changes the view of the plane, as if it was seen from the
        gasInlet (-Y)
372     colormap jet; caxis([0 1]); colorbar
373
374     %Subplots midPoint alpha (Usg, Lb, fb)
375     figure(7)
376     subplot(6, 1, 1);
377     plot(s3.time, s3.alphaMidP);
378     title("Bubbles at Surface 3 mm (midPoint)","FontSize",14); ylabel("Air
        Fraction","FontSize",12);
379     subplot(6, 1, 2);
380     plot(s4.time, s4.alphaMidP);
381     title("Bubbles at Surface 4 mm (midPoint)","FontSize",14); ylabel("Air
        Fraction","FontSize",12);
382     subplot(6, 1, 3);
383     plot(s6.time, s6.alphaMidP);
384     title("Bubbles at Surface 6 mm (midPoint)","FontSize",14); ylabel("Air
        Fraction","FontSize",12);
385     subplot(6,1,4);
386     plot(s7.time, s7.alphaMidP);
387     title("Bubbles at Surface 7 mm (midPoint)","FontSize",14); ylabel("Air
        Fraction","FontSize",12);
388     subplot(6,1,5);
389     plot(s8.time, s8.alphaMidP);
390     title("Bubbles at Surface 8 mm (midPoint)","FontSize",14); ylabel("Air
        Fraction","FontSize",12); ylim([0 1.1]);
391     subplot(6,1,6);
392     plot(s9.time, s9.alphaMidP);
393     title("Bubbles at Surface 9 mm (midPoint)","FontSize",14); ylabel("Air
        Fraction","FontSize",12);
394     xlabel("Time(s)","FontSize",12);
```

```
395
396 %Plot of the average alpha per surface
397 figure(8)
398 subplot(6, 1, 1);
399 plot(s3.realTime, s3.meanAlpha);
400 title("Mean air fraction at Surface 3 mm","FontSize",14); ylabel("Air
      Fraction","FontSize",12); ylim([0 1]);
401 subplot(6, 1, 2);
402 plot(s4.realTime, s4.meanAlpha);
403 title("Mean air fraction at Surface 4 mm","FontSize",14); ylabel("Air
      Fraction","FontSize",12); ylim([0 1]);
404 subplot(6, 1, 3);
405 plot(s6.realTime, s6.meanAlpha);
406 title("Mean air fraction at Surface 6 mm","FontSize",14); ylabel("Air
      Fraction","FontSize",12); ylim([0 1]);
407 subplot(6,1,4);
408 plot(s7.realTime, s7.meanAlpha);
409 title("Mean air fraction at Surface 7 mm","FontSize",14); ylabel("Air
      Fraction","FontSize",12); ylim([0 1]);
410 subplot(6,1,5);
411 plot(s8.realTime, s8.meanAlpha);
412 title("Mean air fraction at Surface 8 mm","FontSize",14); ylabel("Air
      Fraction","FontSize",12); ylim([0 1]); xlabel("Time(s)");
413 subplot(6,1,6);
414 plot(s9.realTime, s9.meanAlpha);
415 title("Mean air fraction at Surface 9 mm","FontSize",14); xlabel("Time(s)");
      ylabel("Air Fraction","FontSize",12); ylim([0 1]);
416
417 save("s3FVcomb1.mat", "s3");
418 save("s4FVcomb1.mat", "s4");
419 save("s6FVcomb1.mat", "s6");
420 save("s7FVcomb1.mat", "s7");
421 save("s8FVcomb1.mat", "s8");
422 save("s9FVcomb1.mat", "s9");
423
424 %% SECTION 2
425 %Run for each mesh as well.
426 %We load the structs created previously
427 clear all
428 load("s7FVcomb1.mat", "s7");
429 %We generate a smaller struct for initial and final bubble points in a
      surface
430 iniFin7 = struct("iniArea", [], "timeIA", [], "finArea", [], "timeFA", []);
431
432 %SEARCH IN s7 OF POINTS
433 %Mean surface alpha
434 i=1;
435 k=1;
436 initial=true;
437 thr=0.0025; %minimum value of alpha considering a bubble
438 while(i<=length(s7.alphaMidP))
439   done=false;
440   if(s7.alphaMidP(i)>=thr&&initial==true)
441     points(k)=i;
442     k=k+1;
443     initial=false;
444     done=true;
445   end
```

```
    if(s7.alphaMidP(i)<=thr&&initial==false&&done==false)
      points(k)=i;
      k=k+1;
      initial=true;
    end
    i=i+1;
  end
  % Remove odd numbers:
  if(rem(length(points),2)==1)
    points(k-1)=[];
  end
  % Initial and final points must be distinguished
  i=1;
  k=1;
  while(i<=length(points))
    iniFin7.iniArea(k)=points(i);
    iniFin7.timeIA(k) = s7.time(points(i));
    i=i+1;
    iniFin7.finArea(k)=points(i);
    iniFin7.timeFA(k) = s7.time(points(i));
    i=i+1;
    k=k+1;
  end
  % Bubble area:
  i=1;
  while(i<=length(iniFin7.iniArea))
    [x, ini] = find(s7.realTime == iniFin7.timeIA(i));
    [y, fin] = find(s7.realTime == iniFin7.timeFA(i));
    aa(i)=trapz(s7.realTime(x:y),s7.meanAlpha(x:y));
    AgA(i)=aa(i)*0.001*0.001;
  i=i+1;
  end
  % Remove little bubbles (simulation errors):
  i=1;
  while(i<=length(AgA))
    if(AgA(i)<=10e-11)
      iniFin7.finArea(i)=[];
      iniFin7.timeFA(i) = [];
      iniFin7.iniArea(i)=[];
      iniFin7.timeIA(i) = [];
      AgA(i)=[];
      aa(i)=[];
      i=i-1;
    end
    i=i+1;
  end

  % save("Ag875FV.mat", "AgA"); %Stable speed is v78, so Area used will be the
    most stable one, in x = 8mm
  save("iniFin7FVcomb1.mat", "iniFin7");
```

E.2. Mesh convergence code

With the *structs* of initial and final bubble points and the area of each one, we compute the frequency, velocity, length and volume of the bubble per mesh. The coarse one with

145k cells is called ”L” (”L” from Light), the medium with 290k cells is called ”M”, and the fine mesh with 435k cells is called ”D” (”D” from Dense) for distinction. The correspondent plots are also made in this file.

```
%COMPUTATIONS AND MESH VALIDATION
%From lighter mesh
load("iniFin3LFV.mat", "iniFin3"); load("iniFin4LFV.mat", "iniFin4"); load("
  iniFin6LFV.mat", "iniFin6"); load("iniFin7LFV.mat", "iniFin7"); load("
  iniFin8LFV.mat", "iniFin8"); load("iniFin9LFV.mat", "iniFin9"); load("
  Ag8LFV.mat", "AgA"); load("s7LFV.mat", "s7"); load("s8LFV.mat", "s8");
iniFin3L = iniFin3; iniFin4L = iniFin4; iniFin6L = iniFin6; iniFin7L =
  iniFin7; iniFin8L = iniFin8; iniFin9L = iniFin9; AgL = AgA; s7L = s7; s8L =
   s8;
%From denser mesh
load("iniFin3DFV.mat", "iniFin3"); load("iniFin4DFV.mat", "iniFin4");
load("iniFin6DFV.mat", "iniFin6"); load("iniFin7DFV.mat", "iniFin7"); load("
  iniFin8DFV.mat", "iniFin8"); load("iniFin9DFV.mat", "iniFin9"); load("
  Ag8DFV.mat", "AgA"); load("s7DFV.mat", "s7"); load("s8LFV.mat", "s8");
iniFin3D = iniFin3; iniFin4D = iniFin4; iniFin6D = iniFin6; iniFin7D =
  iniFin7; iniFin8D = iniFin8; iniFin9D = iniFin9; AgD = AgA; s7D = s7; s8D =
   s8;
%From medium mEsh
load("iniFin3MFV.mat", "iniFin3"); load("iniFin4MFV.mat", "iniFin4"); load("
  iniFin6MFV.mat", "iniFin6"); load("iniFin7MFV.mat", "iniFin7"); load("
  iniFin8MFV.mat", "iniFin8"); load("iniFin9MFV.mat", "iniFin9"); load("
  Ag8MFV.mat", "AgA"); load("s7MFV.mat", "s7"); load("s8LFV.mat", "s8");
Ag = AgA;

%Computations: freq
for i = 1:1:(length(iniFin8L.timeIA)-1) %At stable surface
%We assume that the lengths of each vector are equal, since the process
%does not change.
  freqBL(i) = 1/(iniFin8L.timeIA(i+1)-iniFin8L.timeIA(i));
end
for i = 1:1:(length(iniFin8.timeIA)-1)
  freqB(i) = 1/(iniFin8.timeIA(i+1)-iniFin8.timeIA(i)); %At 8, we have the
  most stable bubble
end
for i = 1:1:(length(iniFin8D.timeIA)-1)
  freqBD(i) = 1/(iniFin8D.timeIA(i+1)-iniFin8D.timeIA(i));
end

%ERROR(%): D mesh correct as it is closer to the real result
errfL = abs(((freqBL(8)-freqBD(8))/freqBD(8))*100);
errf = abs(((freqB(8)-freqBD(8))/freqBD(8))*100);

%Computations: speeds. Different fors as each pair of surfaces has one that
%can be more restrictive (simulation might end before crossing all
%surfaces)
for i = 1:1:length(iniFin4L.iniArea)
  vB34L(i) = (0.004-0.003)/(iniFin4L.timeIA(i)-iniFin3L.timeIA(i));
end
for i = 1:1:length(iniFin4.iniArea)
  vB34(i) = (0.004-0.003)/(iniFin4.timeIA(i)-iniFin3.timeIA(i));
end
for i = 1:1:length(iniFin4D.iniArea)
  vB34D(i) = (0.004-0.003)/(iniFin4D.timeIA(i)-iniFin3D.timeIA(i));
end
```

```
42
43  for i = 1:1:length(iniFin7L.iniArea)
44    vB67L(i) = (0.007-0.006)/(iniFin7L.timeIA(i)-iniFin6L.timeIA(i));
45  end
46  for i = 1:1:length(iniFin7.iniArea)
47    vB67(i) = (0.007-0.006)/(iniFin7.timeIA(i)-iniFin6.timeIA(i));
48  end
49  for i = 1:1:length(iniFin7D.iniArea)
50    vB67D(i) = (0.007-0.006)/(iniFin7D.timeIA(i)-iniFin6D.timeIA(i));
51  end
52
53  %With the most stable surface, we compute the length of the bubble and
54  %its volume at it most stable state (vB78).
55  for i = 1:1:length(iniFin8L.iniArea) %Stable speeds
56    vB78L(i) = (0.008-0.007)/(iniFin8L.timeIA(i)-iniFin7L.timeIA(i));
57    lB8L(i) = (iniFin8L.timeFA(i)-iniFin8L.timeIA(i))*vB78L(i);
58    VolBL(i) = AgL(i)*vB78L(i); %Section multiplied above
59  end
60  for i = 1:1:length(iniFin8.iniArea)
61    vB78(i) = (0.008-0.007)/(iniFin8.timeIA(i)-iniFin7.timeIA(i));
62    lB8(i) = (iniFin8.timeFA(i)-iniFin8.timeIA(i))*vB78(i);
63    VolB(i) = Ag(i)*vB78(i); %Section multiplied above
64  end
65  for i = 1:1:length(iniFin8D.iniArea)
66    vB78D(i) = (0.008-0.007)/(iniFin8D.timeIA(i)-iniFin7D.timeIA(i));
67    lB8D(i) = (iniFin8D.timeFA(i)-iniFin8D.timeIA(i))*vB78D(i);
68    VolBD(i) = AgD(i)*vB78D(i); %Section multiplied above
69  end
70  %ERROR(%)
71  errv78L = ((vB78L(9)-vB78D(9))/vB78D(9))*100;
72  errv78 = ((vB78(9)-vB78D(9))/vB78D(9))*100;
73
74  errlBL = ((lB8L(9) - lB8D(9))/lB8D(9))*100;
75  errlB = ((lB8(9) - lB8D(9))/lB8D(9))*100;
76
77  errvolL = ((VolBL(9)-VolBD(9))/VolBD(9))*100;
78  errvol = ((VolB(9)-VolBD(9))/VolBD(9))*100;
79
80  for i = 1:1:length(iniFin9L.iniArea) %Very unstable because of the outlet,
      but good to see the evolution
81    vB89L(i) = (0.009-0.008)/(iniFin9L.timeIA(i)-iniFin8L.timeIA(i));
82  end
83  for i = 1:1:length(iniFin9.iniArea)
84    vB89(i) = (0.009-0.008)/(iniFin9.timeIA(i)-iniFin8.timeIA(i));
85  end
86  for i = 1:1:length(iniFin8D.iniArea)
87    vB89D(i) = (0.009-0.008)/(iniFin9D.timeIA(i)-iniFin8D.timeIA(i));
88  end
89
90  %MESH CONVERGENCE THROUGH SPEED AND SPACE (V vs X).
91  X = [4, 7, 8, 9]; %X(1) - 34 region. X(2) - 67 region. X(3) - 78 region. X(4)
       - 89 region.
92  % vL = [vB34L(10), vB67L(10), vB78L(10), vB89L(10)];
93  v = [vB34(10), vB67(10), vB78(10), vB89(10)];
94  % vD = [vB34D(10), vB67D(10), vB78D(10), vB89D(10)];
95  figure(1)
96  % plot(X, vL,'-o');
97  hold on;
```

```
plot(X, v,'-o');
hold on;
% plot(X, vD,'-o');
hold off;
% legend("Coarse Mesh", "Medium Mesh", "Fine Mesh");
legend("290,000 elements mesh");
title("Velocity evolution along the pipe"); xlabel("Regions along x-pipe axis (mm)"); ylabel("Velocity(m/s)"); ylim([0.2 0.45]);

%PARAMETERS EVOLUTION IN BUBBLE
figure(2)
subplot(3,1,1);
plot(vB78L, ':k','linewidth',1);
hold on;
plot(vB78, '--k');
hold on;
plot(vB78D, '-k');
title("Stable speed (U78) for each bubble"); ylabel("Velocity (m/s)"); xlabel("Bubble number");xlim([0 9]); ylim([0.35 0.55]);
legend("Coarse mesh (145,000 elements)", "Medium mesh (290,000 elements)", "Fine mesh (435,000 elements)");
subplot(3,1,2);
plot(lB8L, ':k','linewidth',1);
hold on;
plot(lB8, '--k');
hold on;
plot(lB8D,'-k');
title("Bubble's length"); ylabel("Length (m)"); xlabel("Bubble number"); xlim([0 9]);
legend("Coarse mesh (145,000 elements)", "Medium mesh (290,000 elements)", "Fine mesh (435,000 elements)");
subplot(3,1,3);
plot(freqBL,':k', 'linewidth',1);
hold on;
plot(freqB, '--k');
hold on;
plot(freqBD,'-k');
title("Bubbles frequency"); ylabel("Frequency (Hz)"); xlabel("Bubble number");xlim([0 8]); ylim([50 80]);
legend("Coarse mesh (145,000 elements)", "Medium mesh (290,000 elements)", "Fine mesh (435,000 elements)");

figure(3)
subplot(3,1,1)
plot(s7L.time, s7L.alphaMidP);
hold on
plot(s8L.time, s8L.alphaMidP);
title("Alpha at mid point's surface, coarse mesh"); xlim([0 0.16]); xlabel("Time (s)"); ylabel("Air fraction");
legend("Surface 7 mm", "Surface 8 mm");
subplot(3,1,2)
plot(s7.time, s7.alphaMidP);
hold on
plot(s8.time, s8.alphaMidP);
title("Alpha at mid point's surface, medium mesh"); xlabel("Time (s)"); ylabel("Air fraction"); xlim([0 0.16]); ylim([0 1.1]);
legend( "Surface 8 mm");
subplot(3,1,3)
```

```
147   plot(s7D.time, s7D.alphaMidP);
148   hold on
149   plot(s8D.time, s8D.alphaMidP);
150   title("Alpha at mid point's surface, fine mesh"); xlim([0 0.16]); xlabel("
        Time (s)"); ylabel("Air fraction");
151   legend("Surface 7 mm", "Surface 8 mm");
```

E.3. Time step convergence code

The code for time step convergence tests is quite similar to the one for the mesh convergence. One of the few differences is that this one has less plots and uses the letters "T" and "t" to differentiate the bigger time step (T) and the smallest (t).

```
1  %COMPUTATIONS AND TIME VALIDATION
2  clear all
3  %From bigger t.s. (0.00001s)
4  load("iniFin3MTFV.mat", "iniFin3"); load("iniFin4MTFV.mat", "iniFin4"); load("
       iniFin6MTFV.mat", "iniFin6"); load("iniFin7MTFV.mat", "iniFin7"); load("
       iniFin8MTFV.mat", "iniFin8"); load("iniFin9MTFV.mat", "iniFin9"); load("
       Ag8MTFV.mat", "AgA"); load("s7MTFV.mat", "s7"); load("s8MTFV.mat", "s8");
5  iniFin3T = iniFin3; iniFin4T = iniFin4; iniFin6T = iniFin6; iniFin7T = iniFin7;
        iniFin8T = iniFin8; iniFin9T = iniFin9; AgT = AgA; s7T = s7; s8T = s8;
6  %From smaller t.s. (0.0000025s)
7  load("iniFin3MtFVsmall.mat", "iniFin3"); load("iniFin4MtFVsmall.mat", "iniFin4
       ");
8  load("iniFin6MtFVsmall.mat", "iniFin6"); load("iniFin7MtFVsmall.mat", "iniFin7
       "); load("iniFin8MtFVsmall.mat", "iniFin8"); load("iniFin9MtFVsmall.mat", "
       iniFin9"); load("Ag8MtFVsmall.mat", "AgA"); load("s7MtFVsmall.mat", "s7");
       load("s8MtFVsmall.mat", "s8");
9  iniFin3t = iniFin3; iniFin4t = iniFin4; iniFin6t = iniFin6; iniFin7t = iniFin7;
        iniFin8t = iniFin8; iniFin9t = iniFin9; Agt = AgA; s7t = s7; s8t = s8;
10 %From medium t.s. (0.000005s)
11 load("iniFin3MFV.mat", "iniFin3"); load("iniFin4MFV.mat", "iniFin4"); load("
       iniFin6MFV.mat", "iniFin6"); load("iniFin7MFV.mat", "iniFin7"); load("
       iniFin8MFV.mat", "iniFin8"); load("iniFin9MFV.mat", "iniFin9"); load("
       Ag8MFV.mat", "AgA"); load("s7MFV.mat", "s7"); load("s8MFV.mat", "s8");
12
13 %Computations: freq
14 for i = 1:1:(length(iniFin8T.timeIA)-1) %At stable surface
15 %We assume that the lengths of each vector are equal, since the process
16 %does not change.
17   freqBT(i) = 1/(iniFin8T.timeIA(i+1)-iniFin8T.timeIA(i));
18 end
19 for i = 1:1:(length(iniFin8.timeIA)-1)
20   freqB(i) = 1/(iniFin8.timeIA(i+1)-iniFin8.timeIA(i)); %At 8, we have the most
        stable bubble
21 end
22 for i = 1:1:(length(iniFin8t.timeIA)-1)
23   freqBt(i) = 1/(iniFin8t.timeIA(i+1)-iniFin8t.timeIA(i));
24 end
25 %ERROR(%): t t.s. the most convergent result
26 errfT = abs(((freqBT(8)-freqBt(8))/freqBt(8))*100);
27 errf = abs(((freqB(8)-freqBt(8))/freqBt(8))*100);
28
29 %Computations: speeds. Different fors as each pair of surfaces has one that
```

```
%can be more restrictive (simulation might end before crossing all
%surfaces)
for i = 1:1:length(iniFin4T.iniArea)
  vB34T(i) = (0.004-0.003)/(iniFin4T.timeIA(i)-iniFin3T.timeIA(i));
end
for i = 1:1:length(iniFin4.iniArea)
  vB34(i) = (0.004-0.003)/(iniFin4.timeIA(i)-iniFin3.timeIA(i));
end
for i = 1:1:length(iniFin4t.iniArea)
  vB34t(i) = (0.004-0.003)/(iniFin4t.timeIA(i)-iniFin3t.timeIA(i));
end

for i = 1:1:length(iniFin7T.iniArea)
  vB67T(i) = (0.007-0.006)/(iniFin7T.timeIA(i)-iniFin6T.timeIA(i));
end
for i = 1:1:length(iniFin7.iniArea)
  vB67(i) = (0.007-0.006)/(iniFin7.timeIA(i)-iniFin6.timeIA(i));
end
for i = 1:1:length(iniFin7t.iniArea)
  vB67t(i) = (0.007-0.006)/(iniFin7t.timeIA(i)-iniFin6t.timeIA(i));
end

%With the most stable surface, we compute the length of the bubble and
%its volume at it most stable state (vB78).
for i = 1:1:length(iniFin8T.iniArea) %Stable speeds
  vB78T(i) = (0.008-0.007)/(iniFin8T.timeIA(i)-iniFin7T.timeIA(i));
  lB8T(i) = (iniFin8T.timeFA(i)-iniFin8T.timeIA(i))*vB78T(i);
  VolBT(i) = AgT(i)*vB78T(i); %Section multiplied above
end
for i = 1:1:length(iniFin8.iniArea)
  vB78(i) = (0.008-0.007)/(iniFin8.timeIA(i)-iniFin7.timeIA(i));
  lB8(i) = (iniFin8.timeFA(i)-iniFin8.timeIA(i))*vB78(i);
  VolB(i) = AgA(i)*vB78(i); %Section multiplied above
end
for i = 1:1:length(iniFin8t.iniArea)
  vB78t(i) = (0.008-0.007)/(iniFin8t.timeIA(i)-iniFin7t.timeIA(i));
  lB8t(i) = (iniFin8t.timeFA(i)-iniFin8t.timeIA(i))*vB78t(i);
  VolBt(i) = Agt(i)*vB78t(i); %Section multiplied above
end
%ERROR(%)
errv78T = ((vB78T(9)-vB78t(9))/vB78t(9))*100;
errv78 = ((vB78(9)-vB78t(9))/vB78t(9))*100;

errlBT = ((lB8T(9) - lB8t(9))/lB8t(9))*100;
errlB = ((lB8(9) - lB8t(9))/lB8t(9))*100;

errvolT = ((VolBT(9)-VolBt(9))/VolBt(9))*100;
errvol = ((VolB(9)-VolBt(9))/VolBt(9))*100;

for i = 1:1:length(iniFin9T.iniArea) %Very unstable because of the outlet, but
    good to see the evolution
  vB89T(i) = (0.009-0.008)/(iniFin9T.timeIA(i)-iniFin8T.timeIA(i));
end
for i = 1:1:length(iniFin9.iniArea)
  vB89(i) = (0.009-0.008)/(iniFin9.timeIA(i)-iniFin8.timeIA(i));
end
for i = 1:1:length(iniFin8t.iniArea)
  vB89t(i) = (0.009-0.008)/(iniFin9t.timeIA(i)-iniFin8t.timeIA(i));
```

```
end


%PARAMETERS EVOLUTION IN BUBBLE
figure(1)
subplot(3,1,1);
plot(vB78T, ':k','linewidth',1);
hold on;
plot(vB78, '--k');
hold on;
plot(vB78t, '-k');
title("Stable speed (U78) for each bubble"); ylabel("Velocity (m/s)"); xlabel("
    Bubble number");xlim([0 9]); ylim([0.35 0.7]);
legend("Big t.s.(0.00001s)", "Medium t.s.(0.000005s)", "Small t.s.(0.0000025s)
    ");
subplot(3,1,2);
plot(lB8T, ':k','linewidth',1);
hold on;
plot(lB8, '--k');
hold on;
plot(lB8t,'-k');
title("Bubble's length"); ylabel("Length (m)"); xlabel("Bubble number"); xlim
    ([0 9]); ylim([0.0015 0.004]);
legend("Big t.s.(0.00001s)", "Medium t.s.(0.000005s)", "Small t.s.(0.0000025s)
    ");
subplot(3,1,3);
plot(freqBT,':k', 'linewidth',1);
hold on;
plot(freqB, '--k');
hold on;
plot(freqBt,'-k');
title("Bubbles frequency"); ylabel("Frequency (Hz)"); xlabel("Bubble number");
    xlim([0 8]); ylim([50 120]);
legend("Big t.s.(0.00001s)", "Medium t.s.(0.000005s)", "Small t.s.(0.0000025s)
    ");

figure(2)
subplot(3,1,1)
plot(s7T.time, s7T.alphaMidP);
hold on
plot(s8T.time, s8T.alphaMidP);
title("Alpha at mid point's surface, big t.s.(0.00001s)"); xlim([0 0.1175]);
    ylim([0 1]);
legend("Surface 7 mm", "Surface 8 mm");
subplot(3,1,2)
plot(s7.time, s7.alphaMidP);
hold on
plot(s8.time, s8.alphaMidP);
title("Alpha at mid point's surface, medium t.s.(0.000005s)"); xlim([0 0.1175])
    ;
legend("Surface 7 mm", "Surface 8 mm");
subplot(3,1,3)
plot(s7t.time, s7t.alphaMidP);
hold on
plot(s8t.time, s8t.alphaMidP);
title("Alpha at mid point's surface, small t.s.(0.0000025s)"); xlim([0 0.1175])
    ; ylim([0 1]);
legend("Surface 7 mm", "Surface 8 mm");
```

E.4. Contact angle study code

The contact angle code is still similar to mesh and time convergence tests, but it has different plots and there are only computations of the parameters at the the fully developed flux region. If all the convergence tests were in just one file, the code would confuse the user due to an excess of mixed information. That is why all tests are separated and written here. The other option would be changing the inputs and parameter names any time the code had to run, but that is quite slow.

```
%CONTACT ANGLE GRAPHS
clear all
%Contact angle 0
load("s70FV.mat", "s7"); load("s80FV.mat", "s8"); load("iniFin70FV.mat", "
    iniFin7"); load("iniFin80FV.mat", "iniFin8"); load("Ag80FV.mat", "AgA");
s7_0 = s7; s8_0 = s8; iniFin7_0 = iniFin7; iniFin8_0 = iniFin8; Ag_0 = AgA;
%Contact angle 15
load("s715FV.mat", "s7"); load("s815FV.mat", "s8"); load("iniFin715FV.mat", "
    iniFin7"); load("iniFin815FV.mat", "iniFin8"); load("Ag815FV.mat", "AgA");
s7_15 = s7; s8_15 = s8; iniFin7_15 = iniFin7; iniFin8_15 = iniFin8; Ag_15 = AgA
    ;
%Contact angle 30
load("s730FV.mat", "s7"); load("s830FV.mat", "s8"); load("iniFin730FV.mat", "
    iniFin7"); load("iniFin830FV.mat", "iniFin8"); load("Ag830FV.mat", "AgA");
s7_30 = s7; s8_30 = s8; iniFin7_30 = iniFin7; iniFin8_30 = iniFin8; Ag_30 = AgA
    ;
%Contact angle 45
load("s745FV.mat", "s7"); load("s845FV.mat", "s8"); load("iniFin745FV.mat", "
    iniFin7"); load("iniFin845FV.mat", "iniFin8"); load("Ag845FV.mat", "AgA");
s7_45 = s7; s8_45 = s8; iniFin7_45 = iniFin7; iniFin8_45 = iniFin8; Ag_45 = AgA
    ;
%Contact angle 75
load("s775FV.mat", "s7"); load("s875FV.mat", "s8"); load("iniFin775FV.mat", "
    iniFin7"); load("iniFin875FV.mat", "iniFin8"); load("Ag875FV.mat", "AgA");
s7_75 = s7; s8_75 = s8; iniFin7_75 = iniFin7; iniFin8_75 = iniFin8; Ag_75 = AgA
    ;
%Contact angle 90
load("s790FV.mat", "s7"); load("s890FV.mat", "s8"); load("iniFin790FV.mat", "
    iniFin7"); load("iniFin890FV.mat", "iniFin8"); load("Ag890FV.mat", "AgA");
s7_90 = s7; s8_90 = s8; iniFin7_90 = iniFin7; iniFin8_90 = iniFin8; Ag_90 = AgA
    ;
%No contact angle condition
load("iniFin3MFV.mat", "iniFin3"); load("iniFin4MFV.mat", "iniFin4"); load("
    iniFin6MFV.mat", "iniFin6"); load("iniFin7MFV.mat", "iniFin7"); load("
    iniFin8MFV.mat", "iniFin8"); load("iniFin9MFV.mat", "iniFin9"); load("
    Ag8MFV.mat", "AgA"); load("s7MFV.mat", "s7"); load("s8LFV.mat", "s8");
Ag = AgA;

%Stable speed (U78) ang length at surface 8mm
for i = 1:1:length(iniFin8_0.iniArea)
  vB78_0(i) = (0.008-0.007)/(iniFin8_0.timeIA(i)-iniFin7_0.timeIA(i));
  lB8_0(i) = (iniFin8_0.timeFA(i)-iniFin8_0.timeIA(i))*vB78_0(i);
end

for i = 1:1:(length(iniFin8_15.timeIA)-1)
  freqB(i) = 1/(iniFin8_15.timeIA(i+1)-iniFin8_15.timeIA(i));
end
for i = 1:1:length(iniFin8_15.iniArea)
```

```
  vB78_15(i) = (0.008-0.007)/(iniFin8_15.timeIA(i)-iniFin7_15.timeIA(i));
  IB8_15(i) = (iniFin8_15.timeFA(i)-iniFin8_15.timeIA(i))*vB78_15(i);
  VolB(i) = Ag_15(i)*vB78_15(i);
end

for i = 1:1:length(iniFin8_30.iniArea)
  vB78_30(i) = (0.008-0.007)/(iniFin8_30.timeIA(i)-iniFin7_30.timeIA(i));
  IB8_30(i) = (iniFin8_30.timeFA(i)-iniFin8_30.timeIA(i))*vB78_30(i);
end

for i = 1:1:length(iniFin8_45.iniArea)
  vB78_45(i) = (0.008-0.007)/(iniFin8_45.timeIA(i)-iniFin7_45.timeIA(i));
  IB8_45(i) = (iniFin8_45.timeFA(i)-iniFin8_45.timeIA(i))*vB78_45(i);
end

for i = 1:1:length(iniFin8_75.iniArea)
  vB78_75(i) = (0.008-0.007)/(iniFin8_75.timeIA(i)-iniFin7_75.timeIA(i));
  IB8_75(i) = (iniFin8_75.timeFA(i)-iniFin8_75.timeIA(i))*vB78_45(i);
end

for i = 1:1:length(iniFin8_90.iniArea)
  vB78_90(i) = (0.008-0.007)/(iniFin8_90.timeIA(i)-iniFin7_90.timeIA(i));
  IB8_90(i) = (iniFin8_90.timeFA(i)-iniFin8_90.timeIA(i))*vB78_90(i);
end

for i = 1:1:length(iniFin8.iniArea)
  vB78(i) = (0.008-0.007)/(iniFin8.timeIA(i)-iniFin7.timeIA(i));
  IB8(i) = (iniFin8.timeFA(i)-iniFin8.timeIA(i))*vB78(i);
end

%Length VS angle tendency graph
angles = [-10, 0, 15, 30, 45, 75];
l = [IB8(8), IB8_0(8), IB8_15(8), IB8_30(8), IB8_45(8), IB8_75(8)];
v = [vB78(8), vB78_0(8), vB78_15(8), vB78_30(8), vB78_45(8), vB78_75(8)];
figure(1)
plot(angles, l, '-o');
title("8th Bubble length for each contact angle","Fontsize",15); xlabel("
    Contact angle (deg)","Fontsize",12); ylabel("Bubble length (m)","Fontsize
    ",12);
figure(2)
plot(angles, v, '-o');
title("8th Bubble velocity for each contact angle","Fontsize",15); xlabel("
    Contact angle (deg)","Fontsize",12); ylabel("Stable speed (m/s)","Fontsize
    ",12);
```

E.5. Results code

The results code could be mixed with the ones above as well, but to analyse all simulations at once and obtain different plots, this one is also written apart from the rest.

```
%RESULTS: inlet velocities combinations

clear all
%Combination1
```

```
load("s7FVcomb1.mat", "s7"); load("s8FVcomb1.mat", "s8"); load("iniFin7FVcomb1.
    mat", "iniFin7"); load("iniFin8FVcomb1.mat", "iniFin8"); load("Ag8comb1FV.
    mat", "AgA");
s7c1 = s7; s8c1 = s8; iniFin7c1 = iniFin7; iniFin8c1 = iniFin8; Agc1 = AgA;
%Combination2
load("s7FVcomb2.mat", "s7"); load("s8FVcomb2.mat", "s8"); load("iniFin7FVcomb2.
    mat", "iniFin7"); load("iniFin8FVcomb2.mat", "iniFin8"); load("Ag8comb2FV.
    mat", "AgA");
s7c2 = s7; s8c2 = s8; iniFin7c2 = iniFin7; iniFin8c2 = iniFin8; Agc2 = AgA;
%Combination3
load("s7FVcomb3.mat", "s7"); load("s8FVcomb3.mat", "s8"); load("iniFin7FVcomb3.
    mat", "iniFin7"); load("iniFin8FVcomb3.mat", "iniFin8"); load("Ag8comb3FV.
    mat", "AgA");
s7c3 = s7; s8c3 = s8; iniFin7c3 = iniFin7; iniFin8c3 = iniFin8; Agc3 = AgA;
%Combination4
load("s7FVcomb4.mat", "s7"); load("s8FVcomb4.mat", "s8"); load("iniFin7FVcomb4.
    mat", "iniFin7"); load("iniFin8FVcomb4.mat", "iniFin8"); load("Ag8comb4FV.
    mat", "AgA");
s7c4 = s7; s8c4 = s8; iniFin7c4 = iniFin7; iniFin8c4 = iniFin8; Agc4 = AgA;
%Combination5
load("s7MFV.mat", "s7"); load("s8MFV.mat", "s8"); load("iniFin7MFV.mat", "
    iniFin7"); load("iniFin8MFV.mat", "iniFin8"); load("Ag8MFV.mat", "AgA");
s7c5 = s7; s8c5 = s8; iniFin7c5 = iniFin7; iniFin8c5 = iniFin8; Agc5 = AgA;
%Combination6
load("s7FVcomb6.mat", "s7"); load("s8FVcomb6.mat", "s8"); load("iniFin7FVcomb6.
    mat", "iniFin7"); load("iniFin8FVcomb6.mat", "iniFin8"); load("Ag8comb6FV.
    mat", "AgA");
s7c6 = s7; s8c6 = s8; iniFin7c6 = iniFin7; iniFin8c6 = iniFin8; Agc6 = AgA;
%Combination7
load("s7FVcomb7.mat", "s7"); load("s8FVcomb7.mat", "s8"); load("iniFin7FVcomb7.
    mat", "iniFin7"); load("iniFin8FVcomb7.mat", "iniFin8"); load("Ag8comb7FV.
    mat", "AgA");
s7c7 = s7; s8c7 = s8; iniFin7c7 = iniFin7; iniFin8c7 = iniFin8; Agc7 = AgA;
%Combination8
load("s7FVcomb8.mat", "s7"); load("s8FVcomb8.mat", "s8"); load("iniFin7FVcomb8.
    mat", "iniFin7"); load("iniFin8FVcomb8.mat", "iniFin8"); load("Ag8comb8FV.
    mat", "AgA");
s7c8 = s7; s8c8 = s8; iniFin7c8 = iniFin7; iniFin8c8 = iniFin8; Agc8 = AgA;
%Combination9
load("s7FVcomb9.mat", "s7"); load("s8FVcomb9.mat", "s8"); load("iniFin7FVcomb9.
    mat", "iniFin7"); load("iniFin8FVcomb9.mat", "iniFin8"); load("Ag8comb9FV.
    mat", "AgA");
s7c9 = s7; s8c9 = s8; iniFin7c9 = iniFin7; iniFin8c9 = iniFin8; Agc9 = AgA;

%C1 computations
for i = 1:1:(length(iniFin8c1.timeIA)-1)
freqBc1(i) = 1/(iniFin8c1.timeIA(i+1)-iniFin8c1.timeIA(i));
end
for i = 1:1:length(iniFin8c1.iniArea)
vB78c1(i) = (0.008-0.007)/(iniFin8c1.timeIA(i)-iniFin7c1.timeIA(i));
IB8c1(i) = (iniFin8c1.timeFA(i)-iniFin8c1.timeIA(i))*vB78c1(i);
VolBc1(i) = Agc1(i)*vB78c1(i);  %Volume
VolNc1(i) = VolBc1(i)/(0.001*0.001*0.001);   %Normalized volume
end
a0c1 = (0.001*0.001)/VolBc1(7);   %initial slope
%C2 computations
for i = 1:1:(length(iniFin8c2.timeIA)-1)
freqBc2(i) = 1/(iniFin8c2.timeIA(i+1)-iniFin8c2.timeIA(i));
```

```
end
for i = 1:1:length(iniFin8c2.iniArea)
vB78c2(i) = (0.008-0.007)/(iniFin8c2.timeIA(i)-iniFin7c2.timeIA(i));
IB8c2(i) = (iniFin8c2.timeFA(i)-iniFin8c2.timeIA(i))*vB78c2(i);
VolBc2(i) = Agc2(i)*vB78c2(i);  %Volume
VolNc2(i) = VolBc2(i)/(0.001*0.001*0.001);   %Normalized volume
end
a0c2 = (0.001*0.001)/VolBc2(7);   %initial slope
%C3 computations
for i = 1:1:(length(iniFin8c3.timeIA)-1)
freqBc3(i) = 1/(iniFin8c3.timeIA(i+1)-iniFin8c3.timeIA(i));
end
for i = 1:1:length(iniFin8c3.iniArea)
vB78c3(i) = (0.008-0.007)/(iniFin8c3.timeIA(i)-iniFin7c3.timeIA(i));
IB8c3(i) = (iniFin8c3.timeFA(i)-iniFin8c3.timeIA(i))*vB78c3(i);
VolBc3(i) = Agc3(i)*vB78c3(i);  %Volume
VolNc3(i) = VolBc3(i)/(0.001*0.001*0.001);   %Normalized volume
end
a0c3 = (0.001*0.001)/VolBc3(7);   %initial slope
%C4 computations
for i = 1:1:(length(iniFin8c4.timeIA)-1)
freqBc4(i) = 1/(iniFin8c4.timeIA(i+1)-iniFin8c4.timeIA(i));
end
for i = 1:1:length(iniFin8c4.iniArea)
vB78c4(i) = (0.008-0.007)/(iniFin8c4.timeIA(i)-iniFin7c4.timeIA(i));
IB8c4(i) = (iniFin8c4.timeFA(i)-iniFin8c4.timeIA(i))*vB78c4(i);
VolBc4(i) = Agc4(i)*vB78c4(i);  %Volume
VolNc4(i) = VolBc4(i)/(0.001*0.001*0.001);   %Normalized volume
end
a0c4 = (0.001*0.001)/VolBc4(7);   %initial slope
%C5 computations
for i = 1:1:(length(iniFin8c5.timeIA)-1)
freqBc5(i) = 1/(iniFin8c5.timeIA(i+1)-iniFin8c5.timeIA(i));
end
for i = 1:1:length(iniFin8c5.iniArea)
vB78c5(i) = (0.008-0.007)/(iniFin8c5.timeIA(i)-iniFin7c5.timeIA(i));
IB8c5(i) = (iniFin8c5.timeFA(i)-iniFin8c5.timeIA(i))*vB78c5(i);
VolBc5(i) = Agc5(i)*vB78c5(i); %Volume
VolNc5(i) = VolBc5(i)/(0.001*0.001*0.001);   %Normalized volume
end
a0c5 = (0.001*0.001)/VolBc5(7);   %initial slope
%C6 computations
for i = 1:1:(length(iniFin8c6.timeIA)-1)
freqBc6(i) = 1/(iniFin8c6.timeIA(i+1)-iniFin8c6.timeIA(i));
end
for i = 1:1:length(iniFin8c6.iniArea)
vB78c6(i) = (0.008-0.007)/(iniFin8c6.timeIA(i)-iniFin7c6.timeIA(i));
IB8c6(i) = (iniFin8c6.timeFA(i)-iniFin8c6.timeIA(i))*vB78c6(i);
VolBc6(i) = Agc6(i)*vB78c6(i); %Volume
VolNc6(i) = VolBc6(i)/(0.001*0.001*0.001);   %Normalized volume
end
a0c6 = (0.001*0.001)/VolBc6(7);   %initial slope
% %C7 computations
for i = 1:1:(length(iniFin8c7.timeIA)-1)
freqBc7(i) = 1/(iniFin8c7.timeIA(i+1)-iniFin8c7.timeIA(i));
end
for i = 1:1:length(iniFin8c7.iniArea)
vB78c7(i) = (0.008-0.007)/(iniFin8c7.timeIA(i)-iniFin7c7.timeIA(i));
```

```
104 lB8c7(i) = (iniFin8c7.timeFA(i)-iniFin8c7.timeIA(i))*vB78c7(i);
105 VolBc7(i) = Agc7(i)*vB78c7(i);  %Volume
106 VolNc7(i) = VolBc7(i)/(0.001*0.001*0.001);   %Normalized volume
107 end
108 a0c7 = (0.001*0.001)/VolBc7(7);   %initial slope
109 %%C8 computations
110 for i = 1:1:(length(iniFin8c8.timeIA)-1)
111 freqBc8(i) = 1/(iniFin8c8.timeIA(i+1)-iniFin8c8.timeIA(i));
112 end
113 for i = 1:1:length(iniFin8c8.iniArea)
114 vB78c8(i) = (0.008-0.007)/(iniFin8c8.timeIA(i)-iniFin7c8.timeIA(i));
115 lB8c8(i) = (iniFin8c8.timeFA(i)-iniFin8c8.timeIA(i))*vB78c8(i);
116 VolBc8(i) = Agc8(i)*vB78c8(i);  %Volume
117 VolNc8(i) = VolBc8(i)/(0.001*0.001*0.001);   %Normalized volume
118 end
119 a0c8 = (0.001*0.001)/VolBc8(7);   %initial slope
120 %%C9 computations
121 for i = 1:1:(length(iniFin8c9.timeIA)-1)
122 freqBc9(i) = 1/(iniFin8c9.timeIA(i+1)-iniFin8c9.timeIA(i));
123 end
124 for i = 1:1:length(iniFin8c9.iniArea)
125 vB78c9(i) = (0.008-0.007)/(iniFin8c9.timeIA(i)-iniFin7c9.timeIA(i));
126 lB8c9(i) = (iniFin8c9.timeFA(i)-iniFin8c9.timeIA(i))*vB78c9(i);
127 VolBc9(i) = Agc9(i)*vB78c9(i);  %Volume
128 VolNc9(i) = VolBc9(i)/(0.001*0.001*0.001);   %Normalized volume
129 end
130 a0c9 = (0.001*0.001)/VolBc9(7);   %initial slope
131
132 %Vectors for plots from Figure(2) on
133 Usg = [0.05, 0.05, 0.05, 0.1, 0.1, 0.1, 0.5, 0.5, 0.5];
134 Ug = [vB78c1(7), vB78c4(7), vB78c7(7), vB78c2(7), vB78c5(7),  vB78c8(7), vB78c3
        (7), vB78c6(7), vB78c9(7)];
135 Um = [0.1, 0.15, 0.55, 0.15, 0.2, 0.6, 0.55, 0.6, 1];
136 f = [freqBc1(6), freqBc4(6), freqBc7(6), freqBc2(6), freqBc5(6), freqBc8(6),
        freqBc3(6), freqBc6(6), freqBc9(6)];
137 volN = [VolNc1(7), VolNc4(7), VolNc7(7), VolNc2(7), VolNc5(7), VolNc8(7),
        VolNc3(7), VolNc6(7), VolNc9(7)];
138 par = [Usg(1)/(f(1)*0.001), Usg(2)/(f(2)*0.001), Usg(3)/(f(3)*0.001), Usg(4)/(f
        (4)*0.001), Usg(5)/(f(5)*0.001), Usg(6)/(f(6)*0.001), Usg(7)/(f(7)*0.001),
        Usg(8)/(f(8)*0.001), Usg(9)/(f(9)*0.001)];
139 LN = [lB8c1(7)/0.001, lB8c4(7)/0.001, lB8c7(7)/0.001, lB8c2(7)/0.001, lB8c5(7)
        /0.001, lB8c8(7)/0.001, lB8c3(7)/0.001, lB8c6(7)/0.001, lB8c9(7)/0.001];
140 symbols = ["o",  "s", "^","o", "s",  "^", "o", "s", "^"]; %Symbols for plots
        (2), (3), (4)
141 colors = ["b", "r", "k", "b", "r", "k", "b", "r", "k"];
142 %fsat is the maximum frequency out of 3 combinations that share the same
143 %liquid superficial velocity
144 fsatUL1 = max([freqBc1(6), freqBc2(6), freqBc3(6)]);
145 fsatUL2 = max([freqBc4(6), freqBc5(6), freqBc6(6)]);
146 fsatUL3 = max([freqBc7(6), freqBc8(6), freqBc9(6)]);
147 a0UL1 = min([a0c1, a0c2, a0c3]);
148 a0UL2 = min([a0c4, a0c5, a0c6]);
149 a0UL3 = min([a0c7, a0c8, a0c9]);
150 %PARAMETERS EVOLUTION IN BUBBLE
151 figure(1)
152 subplot(3,1,1);
153 plot(vB78c1, "-.b"); hold on; plot(vB78c2, "-.r"); hold on; plot(vB78c3, "-.k")
        ; hold on; plot(vB78c4, "--b"); hold on; plot(vB78c5, "--r"); hold on; plot
```

```
    (vB78c6, "--k"); hold on; plot(vB78c7, "-b"); hold on; plot(vB78c8, "-r");
    hold on; plot(vB78c9, "-k");
154 title("Stable speed (U78) for each bubble"); ylabel("Velocity (m/s)"); xlabel("
    Bubble number");xlim([0 7]);
155 subplot(3,1,2);
156 plot(lB8c1, ":b"); hold on; plot(lB8c2, ":r"); hold on; plot(lB8c3, ":k"); hold
     on; plot(lB8c4, "--b"); hold on; plot(lB8c5, "--r"); hold on; plot(lB8c6,
    "--k"); hold on; plot(lB8c7, "-b"); hold on; plot(lB8c8, "-r"); hold on;
    plot(lB8c9, "-k");
157 title("Bubble's length"); ylabel("Length (m)"); xlabel("Bubble number"); xlim
    ([0 7]);
158 subplot(3,1,3);
159 plot(freqBc1, "b"); hold on; plot(freqBc2, ":r"); hold on; plot(freqBc3, ":k");
     hold on; plot(freqBc4, "--b"); hold on; plot(freqBc5, "--r"); hold on;
    plot(freqBc6, "--k"); hold on; plot(freqBc7, "-b"); hold on; plot(freqBc8,
    "-r"); hold on; plot(freqBc9, "-k");
160 title("Bubbles frequency"); ylabel("Frequency (Hz)"); xlabel("Bubble number");
    xlim([0 6]);
161 legend("Combination 1", "Combination 2", "Combination 3", "Combination 4", "
    Combination 5", "Combination 6", "Combination 7", "Combination 8", "
    Combination 9");
162 %FREQUENCY VS AIR INLET VELOCITY
163 figure(2)
164 for i = 1:1:9
165 scatter(Usg(i), f(i), 100, "filled", strcat(symbols(i), colors(i)));  hold on;
166 end
167 %exponential
168 yUL1 = fsatUL1*(1-exp(-(a0UL1/fsatUL1).*(0:0.01:0.5)));
169 yUL2 = fsatUL2*(1-exp(-(a0UL2/fsatUL2).*(0:0.01:0.5)));
170 yUL3 = fsatUL3*(1-exp(-(a0UL3/fsatUL3).*(0:0.01:0.5)));
171 % plot(0:0.01:0.5,yUL1, "b"); hold on; plot(0:0.01:0.5,yUL2, "r"); hold on;
    plot(0:0.01:0.5,yUL3, "k");
172 title("Frequency VS Air inlet velocity", "Fontsize", 15); xlim([0 0.6]);
173 xlabel("$U_{SG}$ (m/s)", "Interpreter", "Latex", "Fontsize", 15); ylabel("f (Hz
    )", "Fontsize", 15);
174 legend("$U_{SL}$ = 0.05m/s", "$U_{SL} $= 0.1m/s", "$U_{SL}$ = 0.5m/s", "
    Interpreter", "latex", "Fontsize", 15);
175 %NORMALIZED VOLUME VS Usg/(f*side)
176 figure(3)
177 for i = 1:1:9
178 scatter(par(i), volN(i), 100, "filled", strcat(symbols(i), colors(i)));  hold
    on;
179 end
180 l=polyfit(par, volN, 1);
181 volN1 = polyval(l,par);
182 plot(par, volN1);
183 title("Normalized volume VS $\frac{Usg}{f*side}$", "Interpreter", "Latex", "
    Fontsize", 15);
184 xlabel("$\frac{U_{SG}}{f*SectionSide}$", "Interpreter", "Latex", "Fontsize",
    15);
185 ylabel("$\bar{V_{B}}$","Interpreter", "Latex", "Fontsize", 15);
186 legend("$U_{SL}$ = 0.05m/s", "$U_{SL} $= 0.1m/s", "$U_{SL}$ = 0.5m/s", "
    Interpreter", "latex", "Fontsize", 15);
187 %AIR INLET VELOCITY VS MIXED SUPERFICIAL VELOCITY
188 figure(4)
189 for i = 1:1:9
190 scatter(Um(i), Ug(i), 100, "filled", strcat(symbols(i), colors(i)));  hold on;
191 end
```

```
I=polyfit(Um, Ug, 1);
Ug1 = polyval(I,Um);
plot(Um, Ug1);
title("Air inlet velocity VS Mixed superficial velocity", "Fontsize", 15);
xlabel("$U_{M}$ (m/s)", "Interpreter", "Latex", "Fontsize", 15);
ylabel("$U_{SG}$ (m/s)", "Interpreter", "Latex", "Fontsize", 15);
legend("$U_{SL}$ = 0.05m/s", "$U_{SL} $= 0.1m/s", "$U_{SL}$ = 0.5m/s", "
    Interpreter", "latex", "Fontsize", 15);
%NORMALIZED LENGTH VS Usg
figure(5)
for i = 1:1:9
scatter(Usg(i), LN(i), 150, "filled", strcat(symbols(i), colors(i))); hold on;
end
title("Normalized length VS Air inlet velocity", "Fontsize", 25); xlim([0 0.6])
    ;
xlabel("$U_{SG}$ (m/s)", "Interpreter", "Latex", "Fontsize", 25);
ylabel("$\bar{L_{B}}$", "Interpreter", "latex", "Fontsize", 25);
legend("$U_{SL}$ = 0.05m/s", "$U_{SL} $= 0.1m/s", "$U_{SL}$ = 0.5m/s", "
    Interpreter", "latex", "Fontsize", 25);
```

E.6. Pressure analysis code

And lastly, the pressure analysis code organizes the information of the Y sampled surface to visualize the section in a contour plot. This plot justifies the pressure evolution plots as a function of time for a bubble or an entire simulation.

```
%Pressure plot (4 probes)
%Probe0 (0.0005 -0.0005 0.0005) approximately - physical point we desire
%Probe1 (0.0015 -0.0005 0.00125) approximately - physical point we desire
%Probe2 (0.0025 -0.0005 0.0005) approximately - physical point we desire
%Probe3 (0.0015 -0.0005 0.00025) approximately - physical point we desire

%1st: SURFACE Y, SURF REPRESENTATION FOR EACH TIME STEP
clear all
format long; casesList = sort(string(strsplit(ls("/home/gis/bubble15FV/
    postProcessing/surfacesample/"))));
casesList(1) = [];
sY = struct("time", [], "data",struct("x", [], "y", [], "z", [], "alpha", [], "
    dx0", [], "dz0", [], "dx1", [], "dz1", [], "dx2", [], "dz2", [], "dx3", [],
     "dz3", [], "dy", []), "probe0", struct("x", 0, "z", 0, "alpha", 0), "
    probe1", struct("x", 0, "z", 0, "alpha", 0), "probe2", struct("x", 0, "z",
    0, "alpha", 0),"probe3", struct("x", 0, "z", 0, "alpha", 0));
for i=1:1:length(casesList)-1
[fY, err] = fopen("/home/gis/bubble15FV/postProcessing/surfacesample/"+
    casesList(i)+"/alpha.gas_surfaceY.raw", 'r');
if fY == -1
fprintf("========\nCannot open the file fY: error %g, %g\n", err, i);
else
sY.time(i)= double(string(casesList(i)));
n = 1;
while ~feof(fY)
line = fgetl(fY);
words = strsplit(line, ' ');
if strcmp(words(1),'#') == 0 % 0 to be different, 1 to be equal
%First value is at the top of the file. Many values for
```

```
%each time(i) - data(i)
sY.data(i).x(n) = double(string(words(1)));
sY.data(i).y(n) = double(string(words(2)));
sY.data(i).z(n) = double(string(words(3)));
sY.data(i).alpha(n) = double(string(words(4)));
n = n+1;
end
end
fclose(fY);
if(mod(i,1000)==0)
fprintf("File(%g): %s processed successfully\n", i,sY.time(i))
end
end
end
save("SYfile.mat", "sY");
timecontour = input("Insert a number from 1 to 1000 (time increases): ");
[meshCx, meshCz] = meshgrid(unique(sY.data(timecontour).x), unique(sY.data(
    timecontour).z));
F = scatteredInterpolant(sY.data(timecontour).x', sY.data(timecontour).z', sY.
    data(timecontour).alpha');
interpolated_data = F(meshCx, meshCz);
interpolated_data(112:158, 1:1155) = NaN;
interpolated_data(112:158, 2355:11553) = NaN;
figure(9)
set(gcf, "Position", [0, 100, 1000, 200]);
set(gcf, "PaperUnits", "points", "Papersize", [600 800], "PaperPositionMode", "
    auto");
surf(meshCx, meshCz, interpolated_data,'EdgeColor', 'none');
title("Contour plot of Y surface"); xlabel("x axis"); ylabel("z axis");
view(0,90) %changes the view of the plane, as if it was seen from the gasInlet
    (-Y)
colormap jet; caxis([0 1]); colorbar

%Pressure Plots
[pressure_file, err] = fopen("/home/gis/bubble15FV/postProcessing/probes/0/
    p_rgh", 'r');
if err == -1
print("Cannot open the file: error %g", err);
else
disp("Correct, let's continue.");
n=1;
while ~feof(pressure_file)
line = fgetl(pressure_file);
words = strsplit(line, ' ');
if strcmp(words(1),'#') == 0 % 0 to be different, 1 to be equal
time(n) = double(string(words(2)));
probe0(n) = double(string(words(3)));
probe1(n) = double(string(words(4)));
probe2(n) = double(string(words(5)));
probe3(n) = double(string(words(6)));
n = n + 1;
end
end
fclose(pressure_file);
    figure(2)
subplot(2,2,1);
plot(time(2:length(time)), probe0(2:length(probe0)), "b-");
```

```
76 title("Pressure evolution, front point", "Fontsize", 15); xlabel("Time(s)","
       Fontsize", 15); ylabel("Pressure Gage (Pa)","Fontsize", 15); xlim([0.121655
       0.139455]); ylim([-500 2500]);
77 subplot(2,2,2);
78 plot(time(2:length(time)), probe1(2:length(probe1)), "r-");
79 title("Pressure evolution, top point","Fontsize", 15); xlabel("Time(s)","
       Fontsize", 15); ylabel("Pressure Gage (Pa)","Fontsize", 15); xlim([0.121655
       0.139455]); ylim([-500 2500]);
80 subplot(2,2,3);
81 plot(time(2:length(time)), probe2(2:length(probe2)), "k-");
82 title("Pressure evolution, rear point","Fontsize", 15); xlabel("Time(s)","
       Fontsize", 15); ylabel("Pressure Gage (Pa)","Fontsize", 15); xlim([0.121655
       0.139455]); ylim([-500 2500]);
83 subplot(2,2,4);
84 plot(time(2:length(time)), probe3(2:length(probe3)));
85 title("Pressure evolution, bottom point","Fontsize", 15); xlabel("Time(s)","
       Fontsize", 15); ylabel("Pressure Gage (Pa)","Fontsize", 15); xlim([0.121655
       0.139455]); ylim([-500 2500]);
86 figure(3)
87 plot(time(2:length(time)), probe0(2:length(probe0)), "b-"); hold on; plot(time
       (2:length(time)), probe1(2:length(probe1)), "r-");
88 hold on; plot(time(2:length(time)), probe2(2:length(probe2)), "k-"); hold on;
       plot(time(2:length(time)), probe3(2:length(probe3)), "g-");
89 title("Pressure evolution for a bubble","Fontsize", 15); xlabel("Time(s)","
       Fontsize", 15); ylabel("Pressure Gage (Pa)","Fontsize", 15); xlim([0.121655
       0.139455]); ylim([-500 2500]);
90 legend("Front point", "Top point", "Rear point", "Bottom point");
91 end
```